絵とき 8ビットマイコンCPU大研究

●8080、Z80をもっと知りたい人のために
●BASICからアセンブラ、マシン語、CP/Mまで

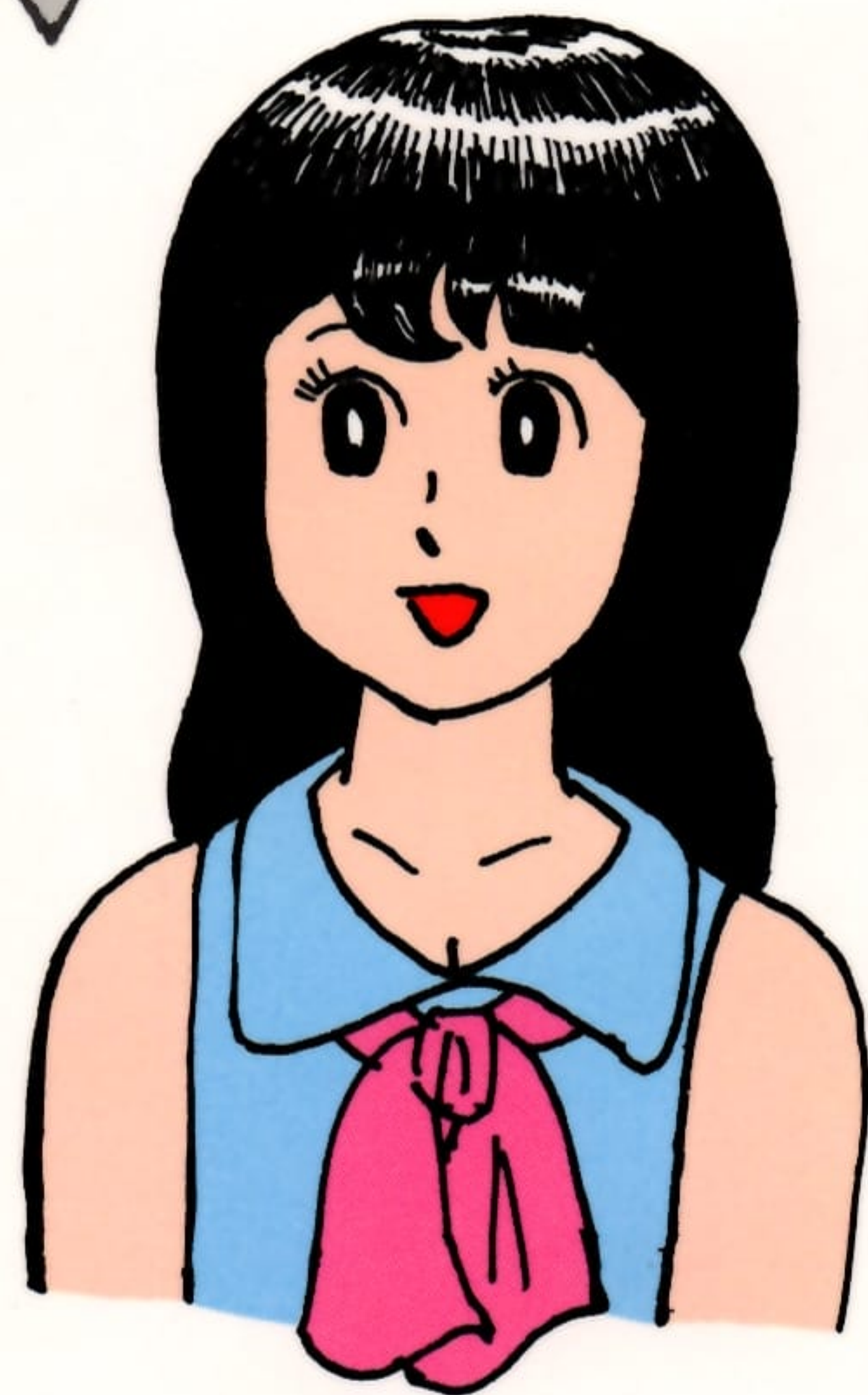
と6809

笹倉 正義

日本実業出版社

8ビットマイコンCPU大研究
笹倉正義
日本実業出版社

962150

*PC、MZ、MSXなど
80系のマイコンユーザー
のみなさん、ぜひこの本を
お読みください。
*FMなど68系のユーザーの
みなさんにも、きっと
お役に立ちます。

友子が絵で解説した
この本を読んで
いただければ……

- ●CPUのことがわかる
- ●マシン語のことがわかる
- ●マイコンの高度な活用がわかる

絵とき 8ビットマイコン CPU大研究

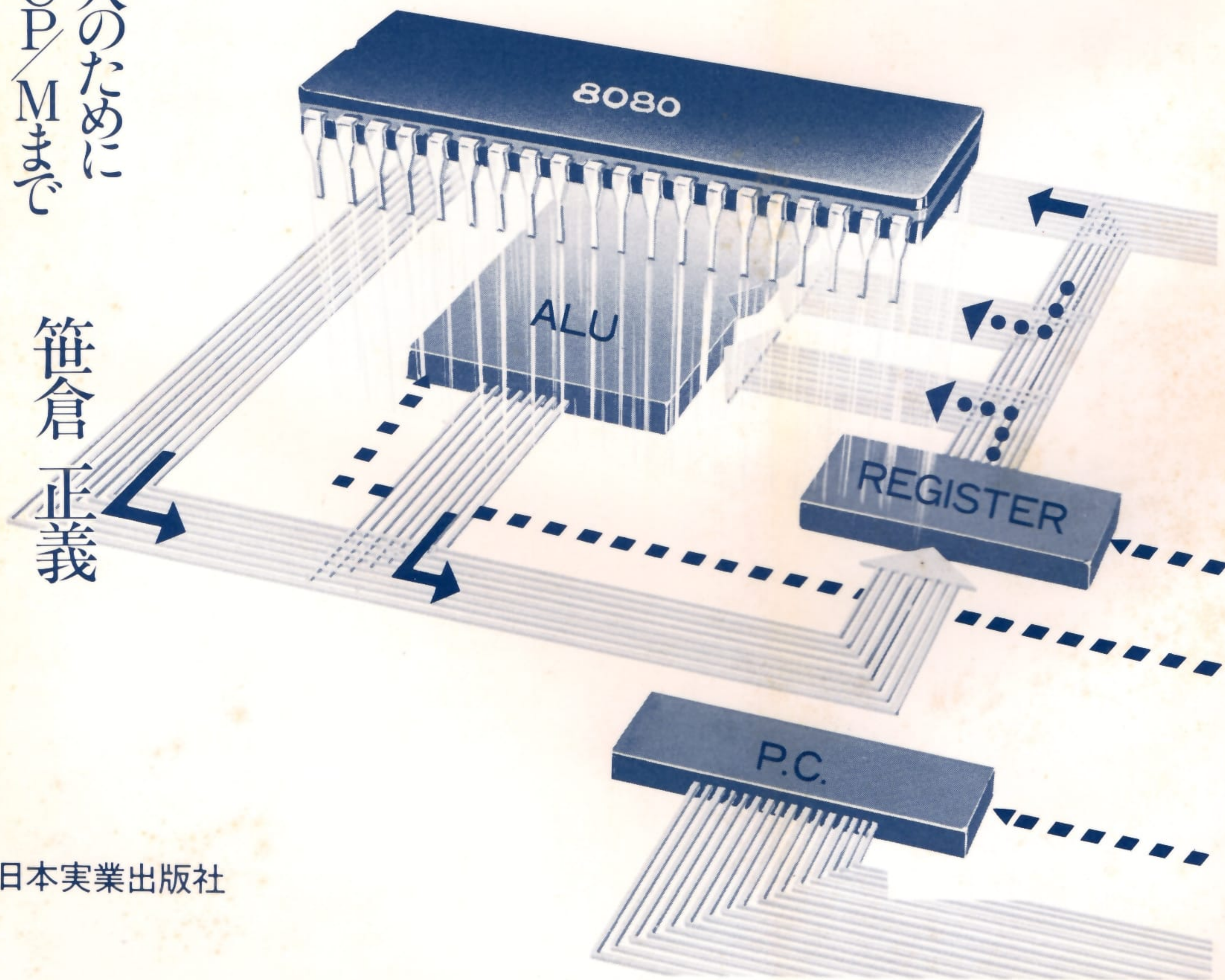
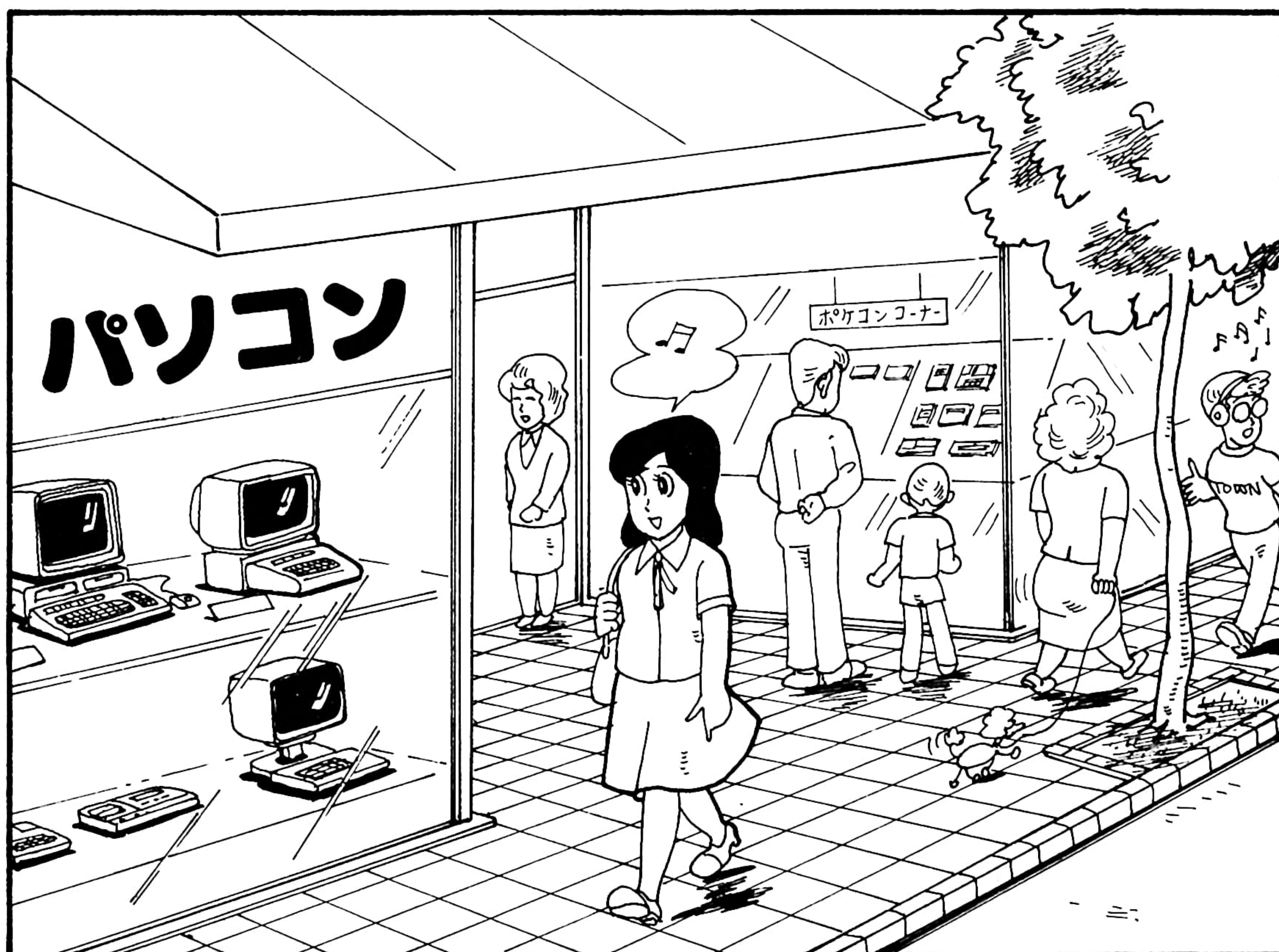
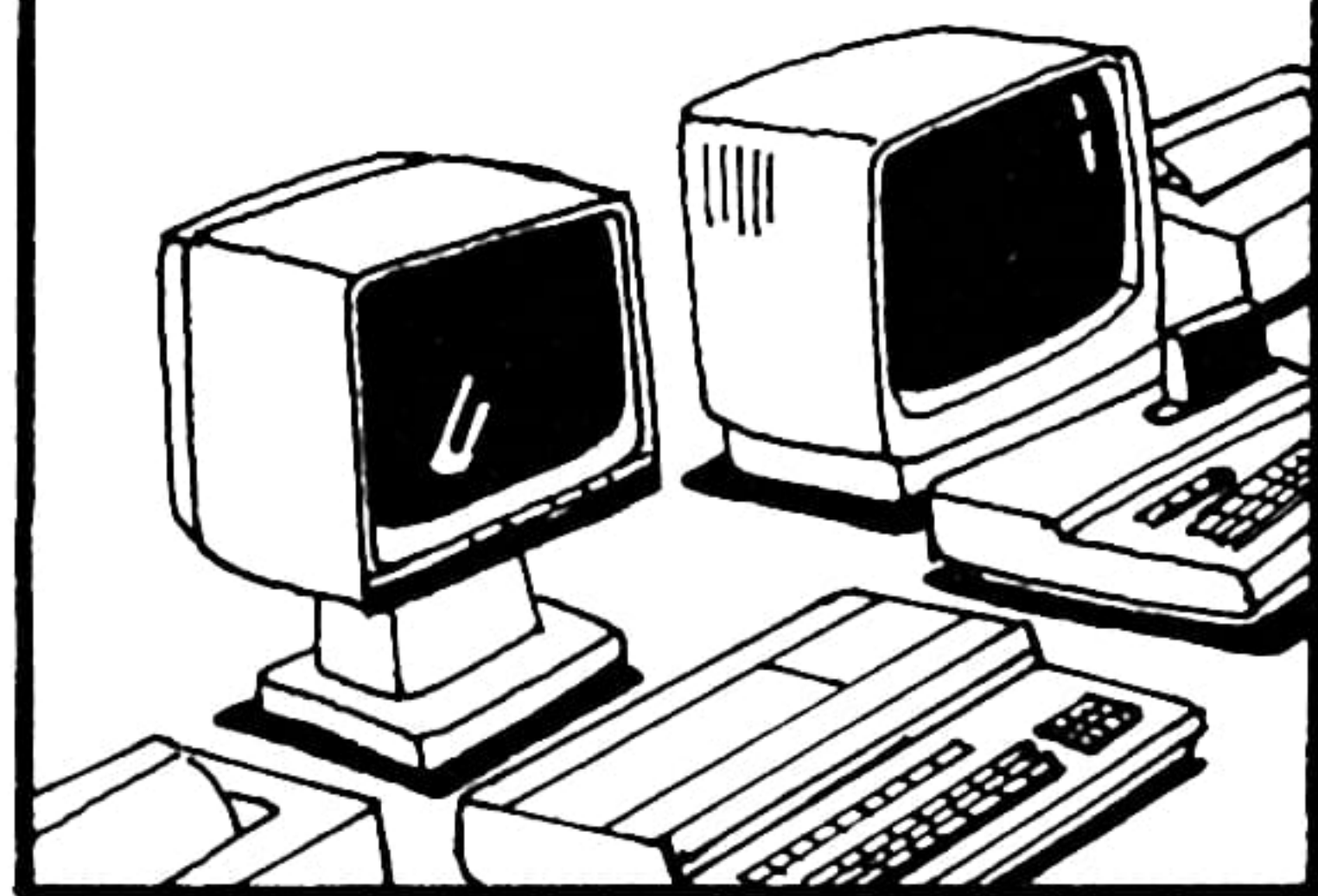
●8080、Z80をもっと知りたい人のために

●BASICからアセンブラ、マシン語、CP/Mまで

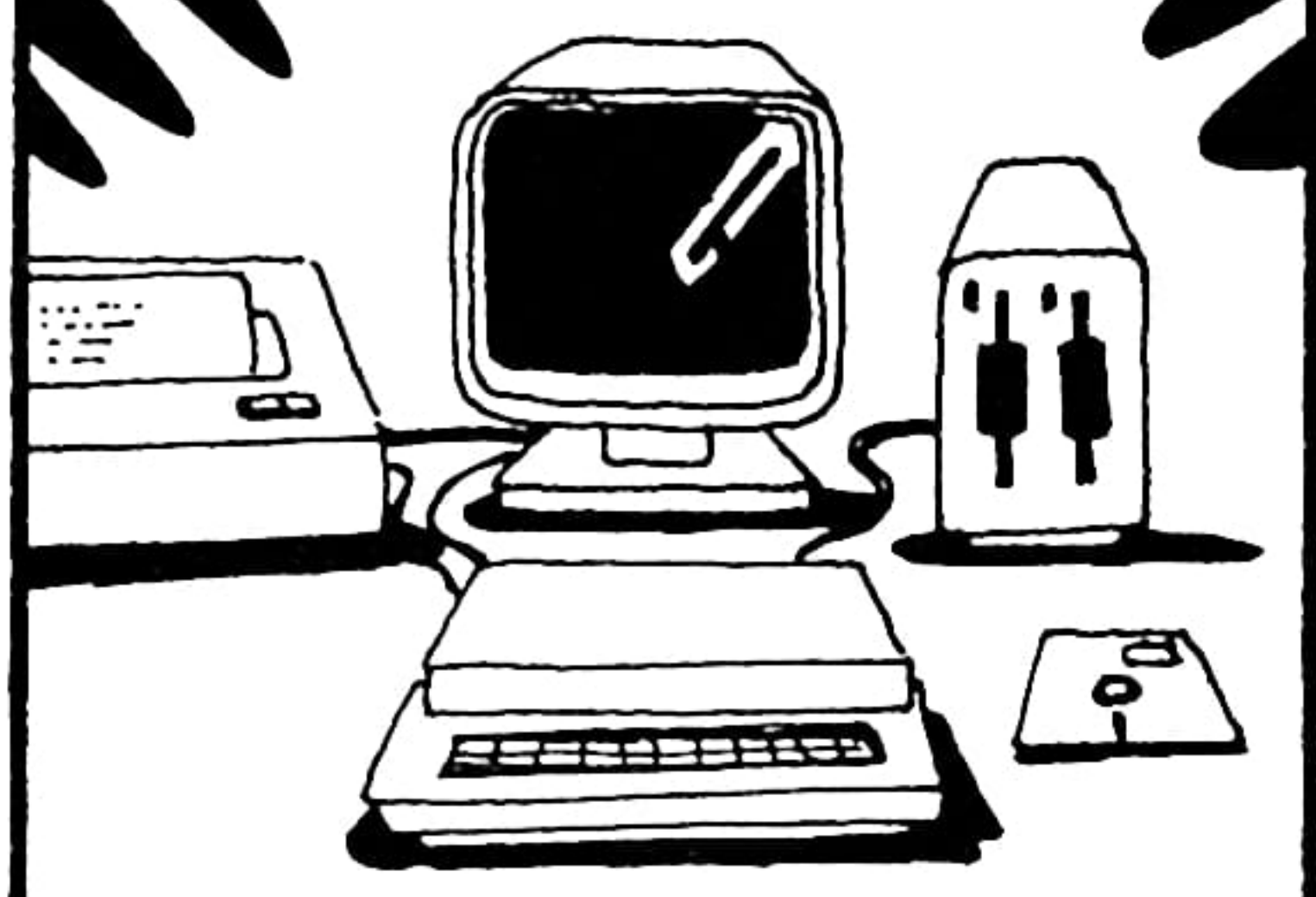
笹倉　正義

日本実業出版社

パソコン

ポケコンコーナー

パソコンは、今や家電メーカーも進出、中学生や小学生まで持っている時代です。

きょうはパソコンのウィンドショッピングです。

あ、はじめまして。私は友子です。どうぞよろしく。

パソコンといってもコンピュータなのですから、そう簡単に自在に使いこなすというわけにはいきません。

ことばだけがCMベースで先行し、中身がついていっていないようなのです。

ベーシック

ラム

シーピーエム

データバス

ロム

ところが、ゲーム以外になにかできるのかというとなかなかみたいです。

車を運転するのに必要なことはまずガソリンを入れること。これがないと車は動きません。

車を運転するのにエンジンのしくみを知っていなければならないかというとそうでもないですものね。

DTTQ

コンピュータを使うのも車と同じで運転の方法さえ教えてもらえば誰だって動かせます。

ブブー

コンピュータもそのうちセンサーが発達すればもっともっと使いやすいようになっていく気がしますがソフト面の開発力が問題です。

ボク友子ちゃんの言葉わかる。

うん

ただし、車を走らせるためには免許が必要ですね。今のところコンピュータには必要ないですけど。

あとは足もとの３つのペダルとハンドルとレバーを適当に操作できれば車は一応動きます。

DTT

ここに取り出しましたのは、マイコンのことを書いたマンガです。

これでマイコンのあらましを勉強してみませんか？

うまく運転して活用するためには、やっぱり本質的なもの、マイクロコンピュータそのものの知識が必要だと思うのです。

コンピュータは車と違って走っているのが見えませんからやっかいです。

RUN…

トットットコ

マイコンの勉強は、まずBASICから入るのが普通です。BASICは人間の言葉に近い表現なので、比較的わかりやすいからです。しかし、BASICだけで使っているうちは、CPUのことはなかなかわかりません。

CPUがいったいどんな働きをどのようにしているのか知りたいと思いませんか。それがわかれば、マシン語がわかってきます。そして、コンピュータの活用の幅がグーンと広がってくるはずです。

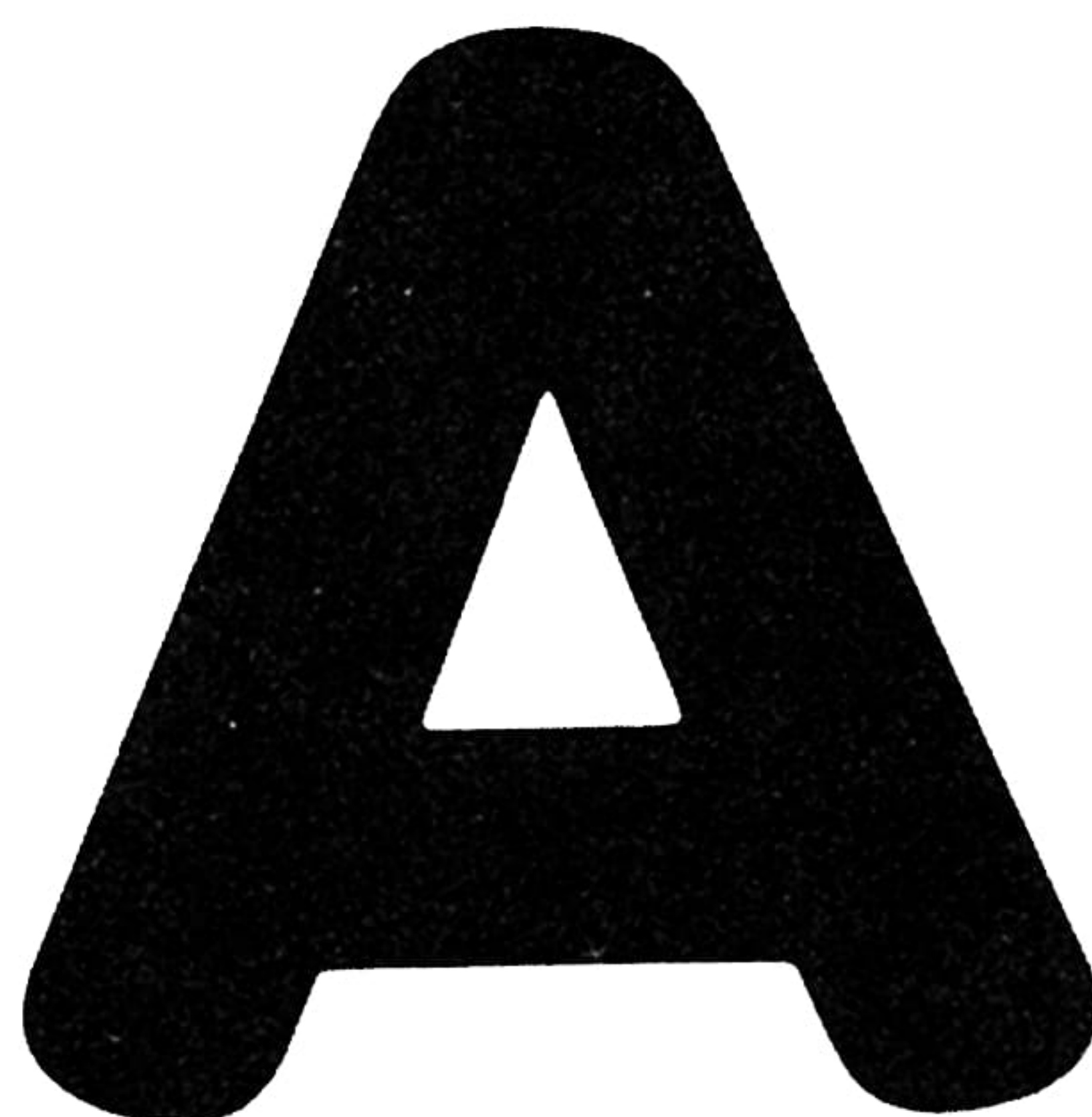
……というようなわけで、友子ちゃんがとり出しましたこの本は、マイコンの入門書としてはちょっと変わっています。一応はマンガの格好をしていますが、面白いマンガではありません。けれども、マイコンのことをもっと知りたい、という人にとっては、本当に役立つ内容を盛り込んであります。

言葉の数を多くして文章でくどくどと説明するのでなく、ポイントをできるだけ簡潔におさえるようにしています。そのため、若干のみ込みにくいところがあるかもしれません。あるいは、あなたがまだマイコンやその本にあまりなじみがない場合、一回ざっと読んだだけではわかりにくいでしょう。ハードに興味のない人たちにとっては、若干退屈なところも出てきます。

しかし、それで放り出さずに、何回か、じっくりと、繰り返して読んでみてください。眺めてみてください。あきたら、しばらく放っておいて、またしばらくして手にとってみましょう。

どうしても読めないところは、どんどんとばしていってください。でも何回かめくっているうちに、だんだんと、わかってきます。わかる箇所が少しずつ増えてきます。これは、そういう本なのです。

＊　＊

さて、かめばかむほど味の出るスルメのようなこの本の内容ですが………

A もくじ

BASICはパソコン理解の第一歩

まず、定石どおりBASICのお勉強から……。

「BASICなんて勉強してみたってしょうがない。これからは簡易言語だ」などという人が多くなっています。それも一理あります。要は、使う人のニーズの問題でしかないでしょう。

BASICも、このA章でやっている程度のことすら理解せず、やろうともせず、そういう論に追随するのはどんなものでしょうか。

ここでは、パソコンを理解する第一歩は、やはりBASICからということで、簡単なBASICの実用知識を整理し、データファイルの操作ができるところまで述べています。

これが完全にわかれば、あとはあなた自身のニーズに応じて、パッとヒラメクものが何かあるはずです。

5

B

マシン語がわかればもっと使いこなせる

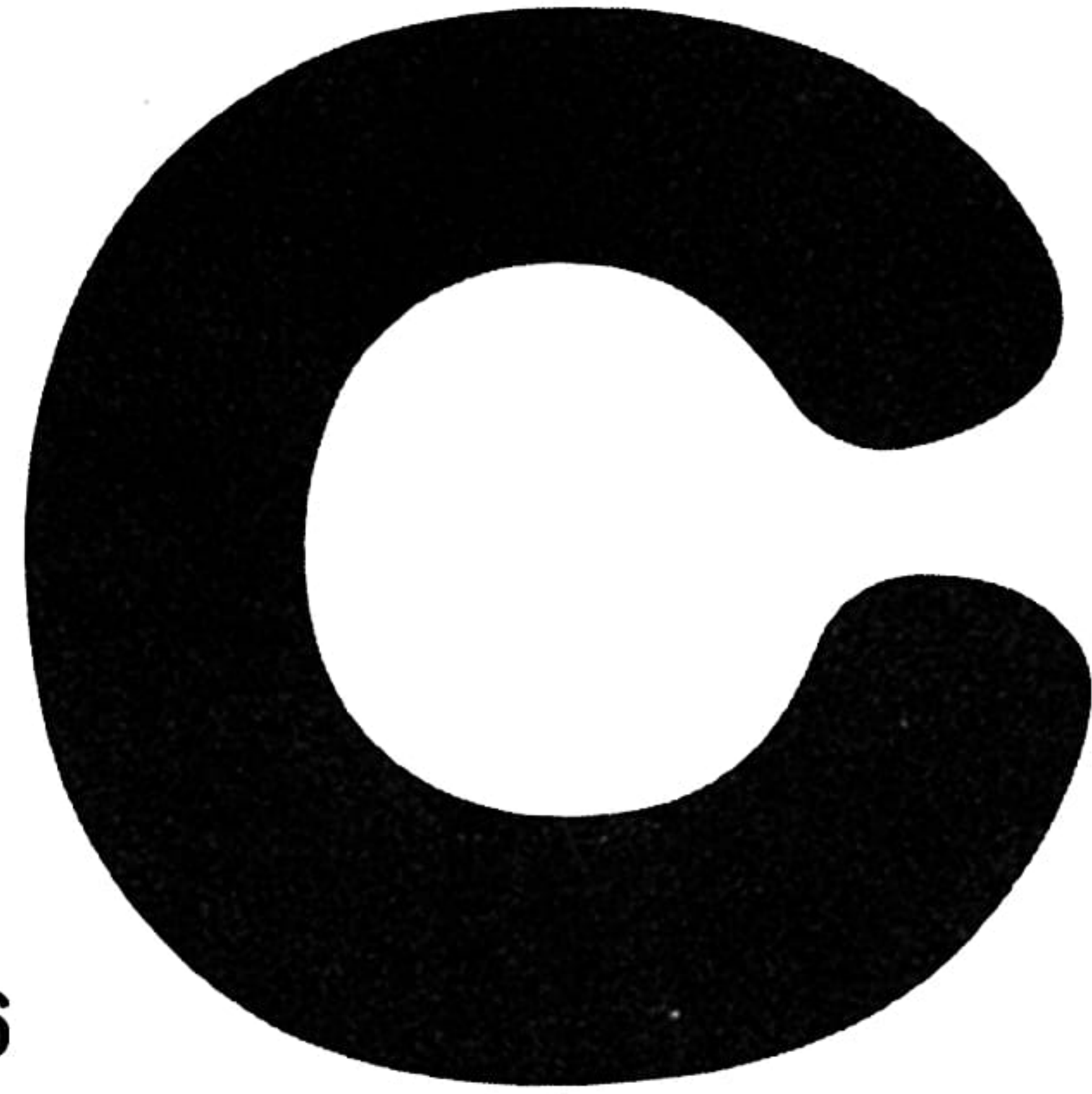
最近の傾向としては、確かに簡易言語派が増えてはいます。けれど、その一方で、「BASICは一応卒業したから、今度はマシン語に挑戦したい」という人々もたくさんいるはずです。

別にマニアというわけではないけれど、同じやるならもう少し突っ込んでみたい、という自然な願望です。

現に、ゲームなどはマシン語を使わないと、あまり面白いものはできません。マイコン誌などに載るプログラムリストも、16進表記のマシン語リストの方が多くなっていますが、ただわけもわからず打ち込んでもつまらないですね。

少しでもマシン語がわかれば、もっともっと使い方が広がってきます。楽しさが増えます。

少々めんどくさいのですが、ほら、例の「知るは喜びなり、と申します」という名文句もあるではありませんか。がんばりましょう。

マシン語はCPUによって異なります。ここでは8080を中心に説明します。それは、8080がわかれば80系のイシは全部わかり、Z80もわかるからです。おまけで6809についてもちょっとふれました。

C CPUの中はどうなっているか

マシン語をやるなら、何かイシを決めてやらないと、何が何だかわからなくなる、といいます。それは、自分がどのパソコンを持っているか、で決まります。でも、マシン語とはどんなものかがわかれば、あとは比較的早いのです。6809の人も、これでずっと入門が楽になりますよ。

それにしても、CPUというのはちっちゃなものなのに、すごいことができるものです。マシン語をよりよく知るためのCPUの中の話を少々……。

このあたり、回路がどうだとか、クロックがどうだとかいう話になり、ちょっとむずかしいかもしれません。まア、だいたいの感じを理解していただければ……。

D

CP／Mとアセンブラの活用知識

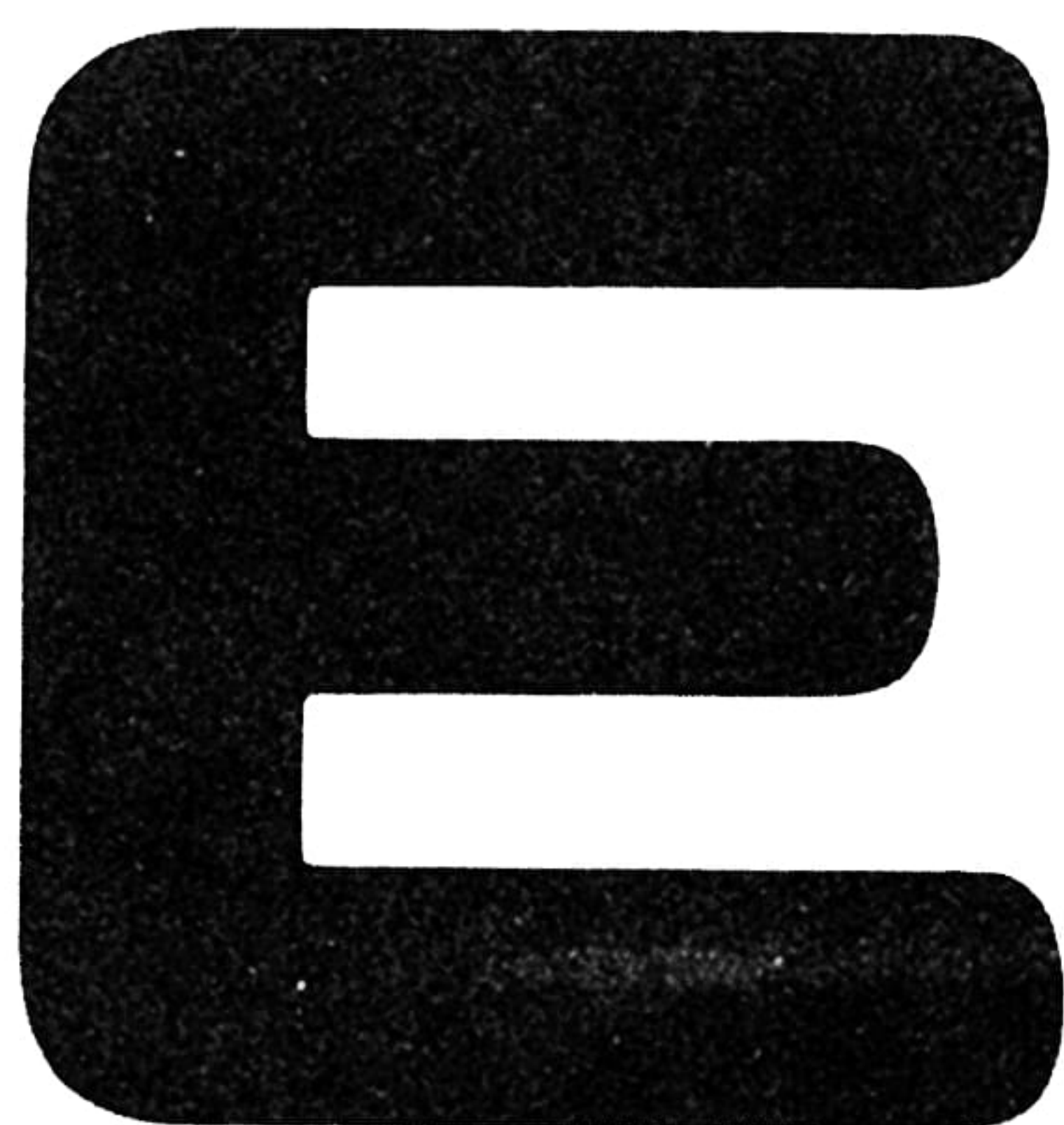
CPUの機能をフルに活かすためのソフトの中心になるのが、OS（オペレーティングシステム）といわれるものです。ディジタルリサーチ社が製品化しているCP／Mとは、このOSの一種なのです。A社のパソコンにはすでにそれ用のOSがあるのですが、そのうえ、さらに標準OSが必要になるというのはどういうことでしょうか。

CP／Mは、本来的には、強力なアセンブラなどを備えています。そしてZ80上で動きます。ですからZ80コンパチマシン（数が多い）のソフト開発がその主要な役割なのですが……。

ニーモニックコードをマシン語に変換するアセンブラは、CP／Mだけにあるわけではありません。簡単なのから複雑なのまで、各機種向けにいろいろあります。

実際にアセンブラを使ってみると、どんなことになるのでしょうか。ただ、言葉の意味を知っているという段階から一歩を踏み込んで、CP／Mやアセンブラなどを研究してみよう、というのがこのD章です。

E

RS—232Cでデータ転送

この頃は、たいていのパソコンにRS—232Cポートとやらいうものがついています。あるいはつけられるようになっています。

ところが、これがいったいどんな役に立ってくれるのかというと、あまりよくわからないようです。

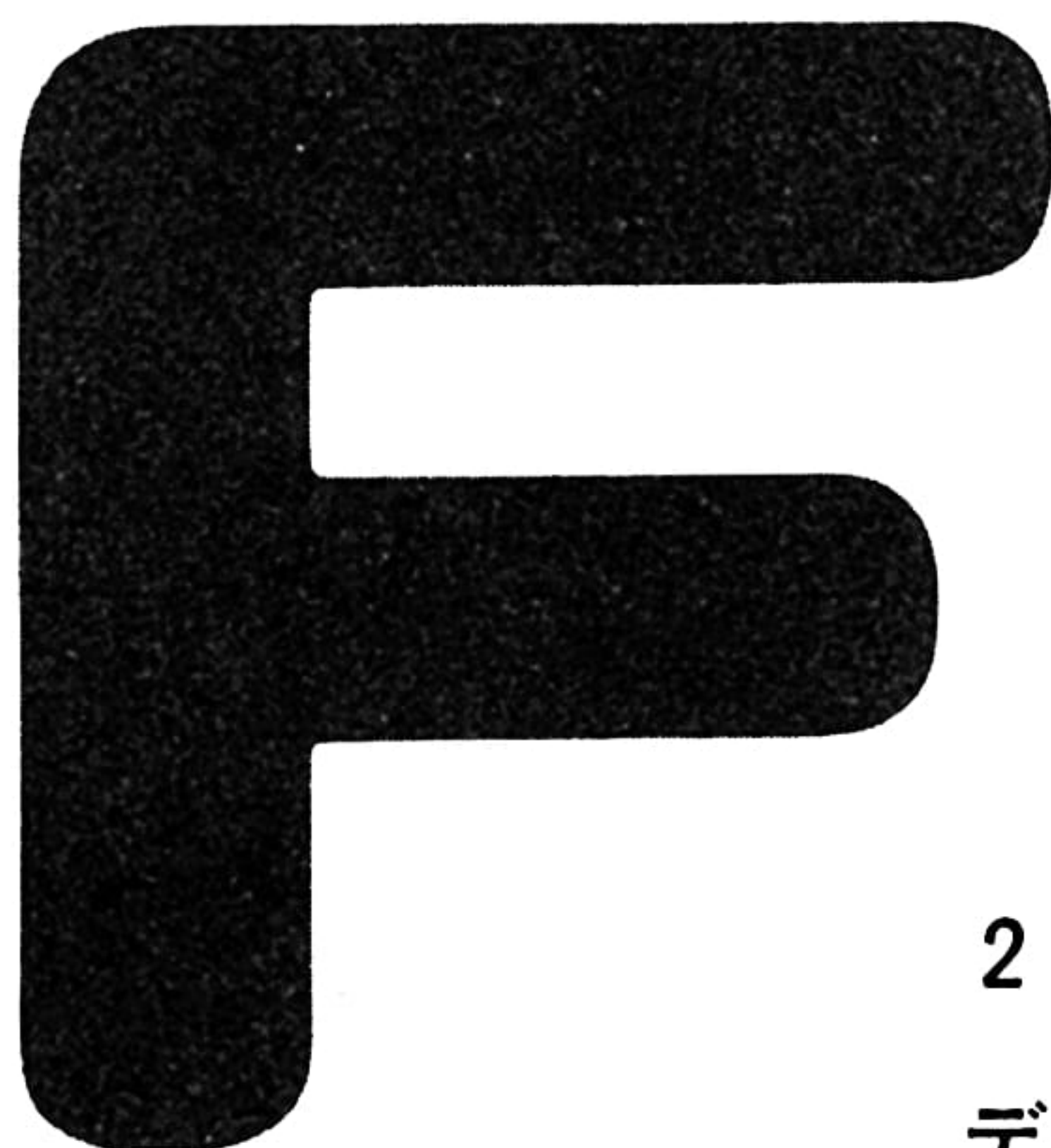
たとえば、二台のパソコンをつないで、データのやりとりができるとしたら、あなたはどんな活用法、使い方を考えるでしょうか。

こういうことができるようになると、マイコン活用も立派なものになってきそうですね。

ここでは、そのRS—232Cを使ってのデータ転送の基本テクニックを紹介していますので、ぜひ実地への応用を工夫してみてください。これはBASICレベルですが、RS—232CのコントロールのためのBASICは、機種により少しずつ異なりますので、注意する必要があります。

F

データ源から直接入力させる法

絵とき＊笹倉正義
カバーデザイン＊井上丈児

二台のパソコンを結んで使うという使い方を、さらに徹底して追究すると、どういうことになるでしょうか。
実際にパソコンを活用しようとして、ソフトをつくり、実地に使うとします。いったい何が問題になると思いますか。結局は人間がやらなければならないデータ入力でしょう。
そこで、もうちょっとがんばって、データ入力も機械からパソコンへ直接やってしまう方法はないだろうか、と考えました。
この最後のF章では、実際にやっているひとつの例を示しています。これも応用はかなりできるのではないでしょうか。こうなるとハードの工作とか発注とかいう問題も出てきて、お金もかかりますか、これは8ビットマイコンの活用を極めた、ひとつの姿として参考にしていただきたいのです。

……ざっと以上のような構成になっています。一応、初心者から、ある程度経験している人まで、それぞれのレベルに応じて、幅広く役立てていただけるのではないかと思います。

具体的なパソコンもいくつか出てきますが、これはメーカーや機種を限定した本ではありません。どんな機種をお持ちの方にも参考になるはずです。ただし、ＣＰＵは最も広く普及している８０系・Ｚ８０を中心にして説明しています。

本書の中で、みなさんをご案内する友子ちゃん（さん、というべきでしょうか）は、とあるメーカーに勤務しているＯＬです。

彼女は大変な勉強家で、ここ数年にわたり、コツコツとマイコン研究に打ち込んできました。会社の制服らしきものを着て登場することが多いのは、彼女がその研究の成果を認められて、社内の小さな研修会などのインストラクターをやったりしていたからでしょうか。

どうぞ、みなさんも彼女の話を聞いてやってください。

昭和59年8月

著　者

A BASICはパソコン理解の第一歩

‘パソコン’というと‘BASIC’となるのですが、実際には、それぞれのCPUのマシン語で高速に実行しているのです。ここではまずBASICのおさらいをしておいてB章以降でマシン語の研究をしてみたいと思います。

ソフトウェア

インタプリタがプログラムを1行ずつマシン語（電圧のon-off）で実行します。

BASICインタプリタ
（マシン語プログラム）

BASICプログラム
```
10␣INPUT␣A
20␣A=A*A
30␣PRINT␣A
30␣END
```

ハードウェア

CPU
（中央処理装置、マイコンの場合はひとつの小さなチップで演算機能付です。）

RAMメモリ
（プログラムやデータを入れる）

ROMメモリ
（インタプリタやモニタが入っている）

I/Oポート

ディスク

CRT

キーボード

プリンタ

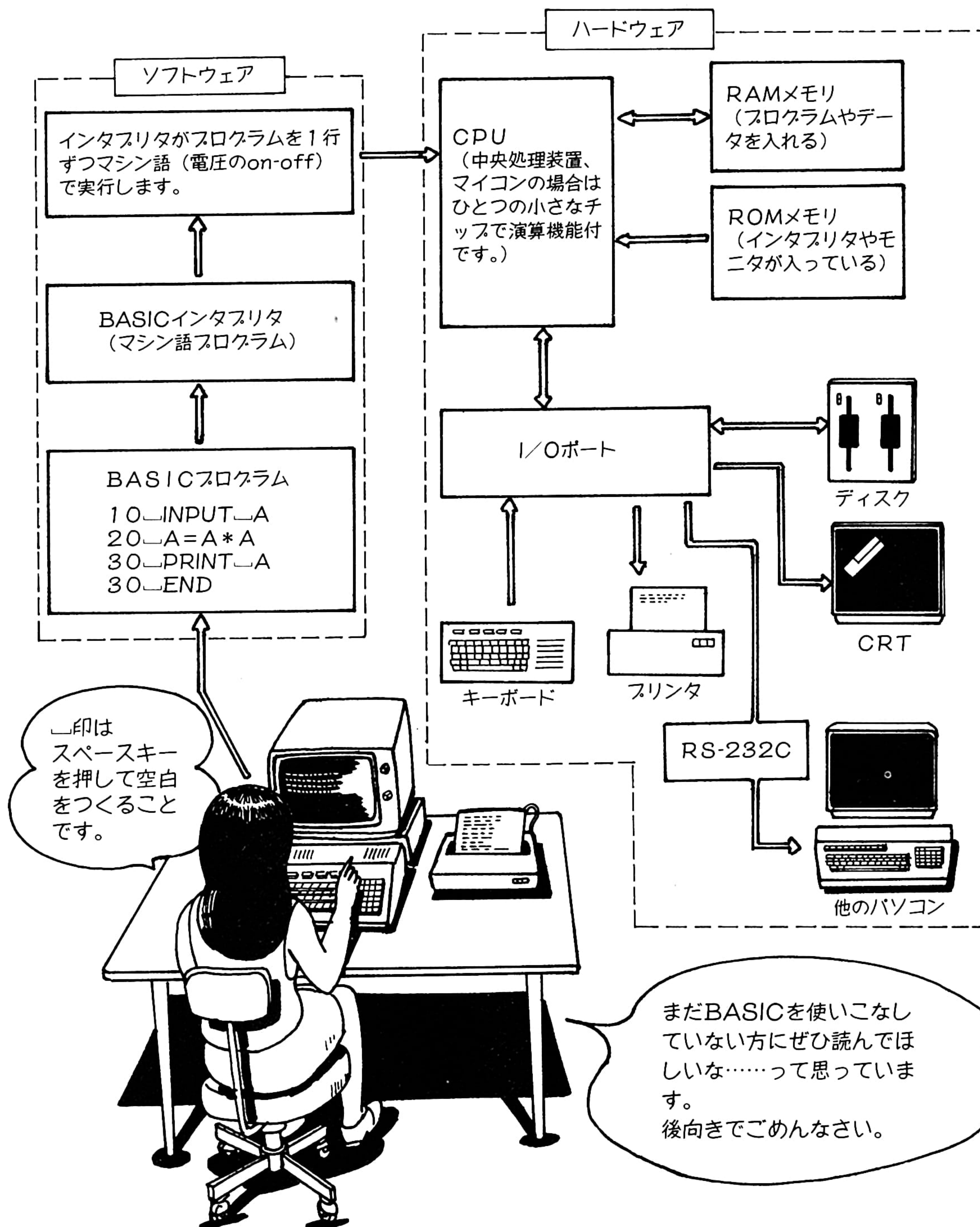
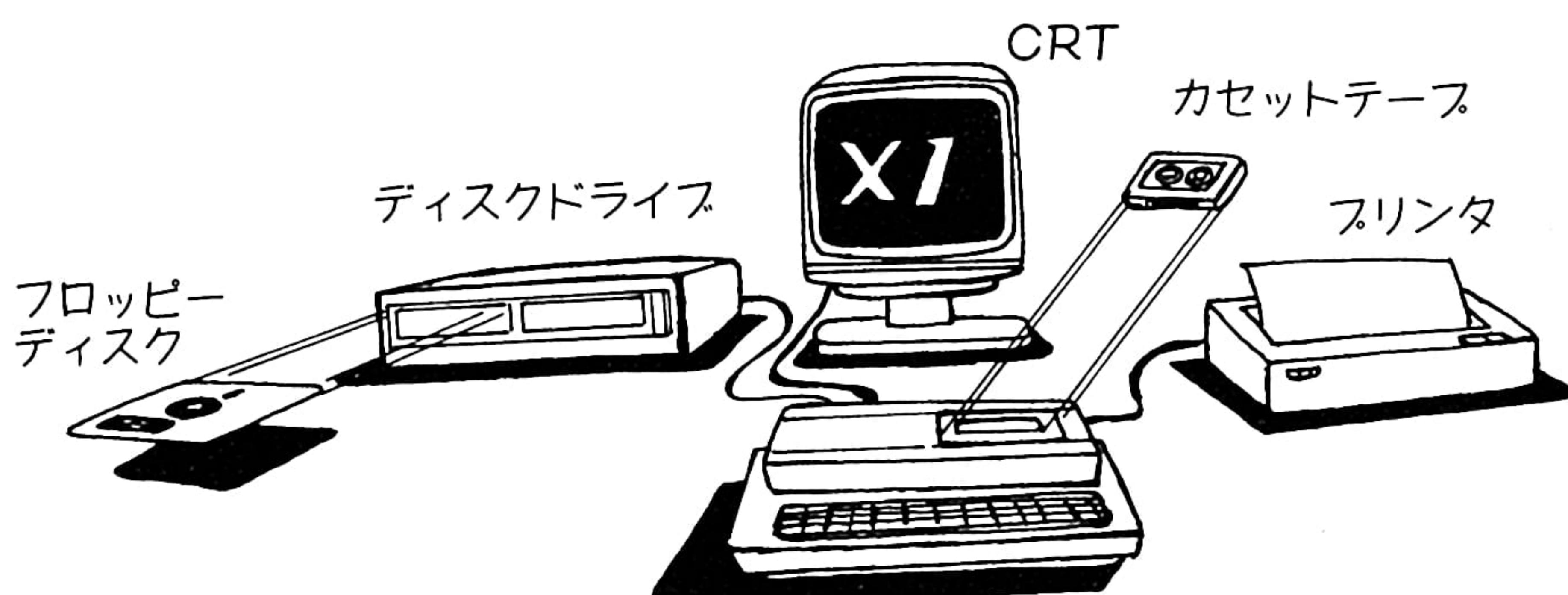
RS-232C

他のパソコン

␣印はスペースキーを押して空白をつくることです。

まだBASICを使いこなしていない方にぜひ読んでほしいな……って思っています。
後向きでごめんなさい。

①パソコンはBASICが使える

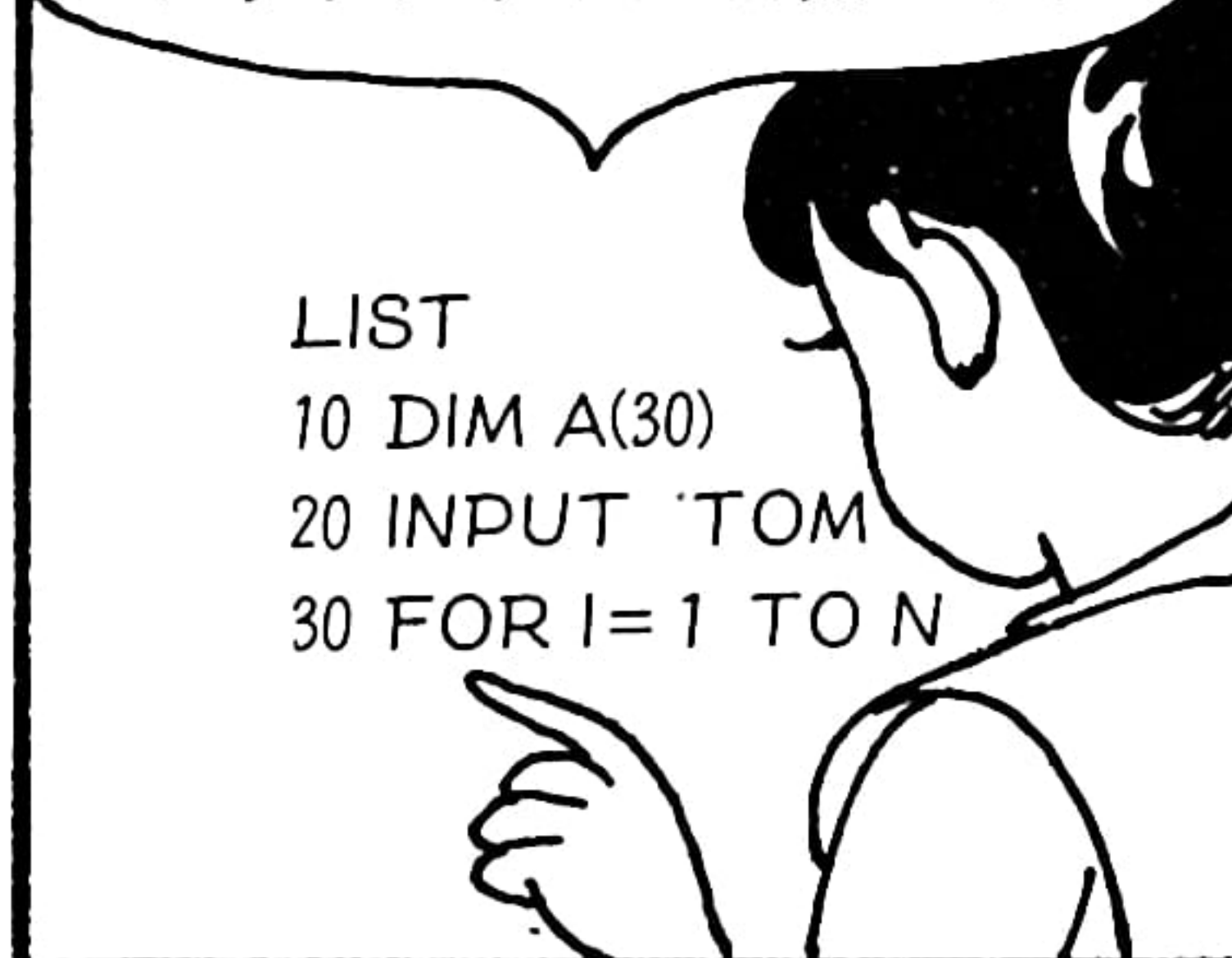
CRT
カセットテープ
ディスクドライブ
プリンタ
フロッピーディスク

ゲームのプログラムはマイコン誌に載っており、そのとおり打ち込めば一応プログラムを走らせることができます。

300 LOCAT
310 LINE

パソコンを買ってまずやるのがゲームです。店頭でもグラフィックやアニメーションでお客の気をひいています。

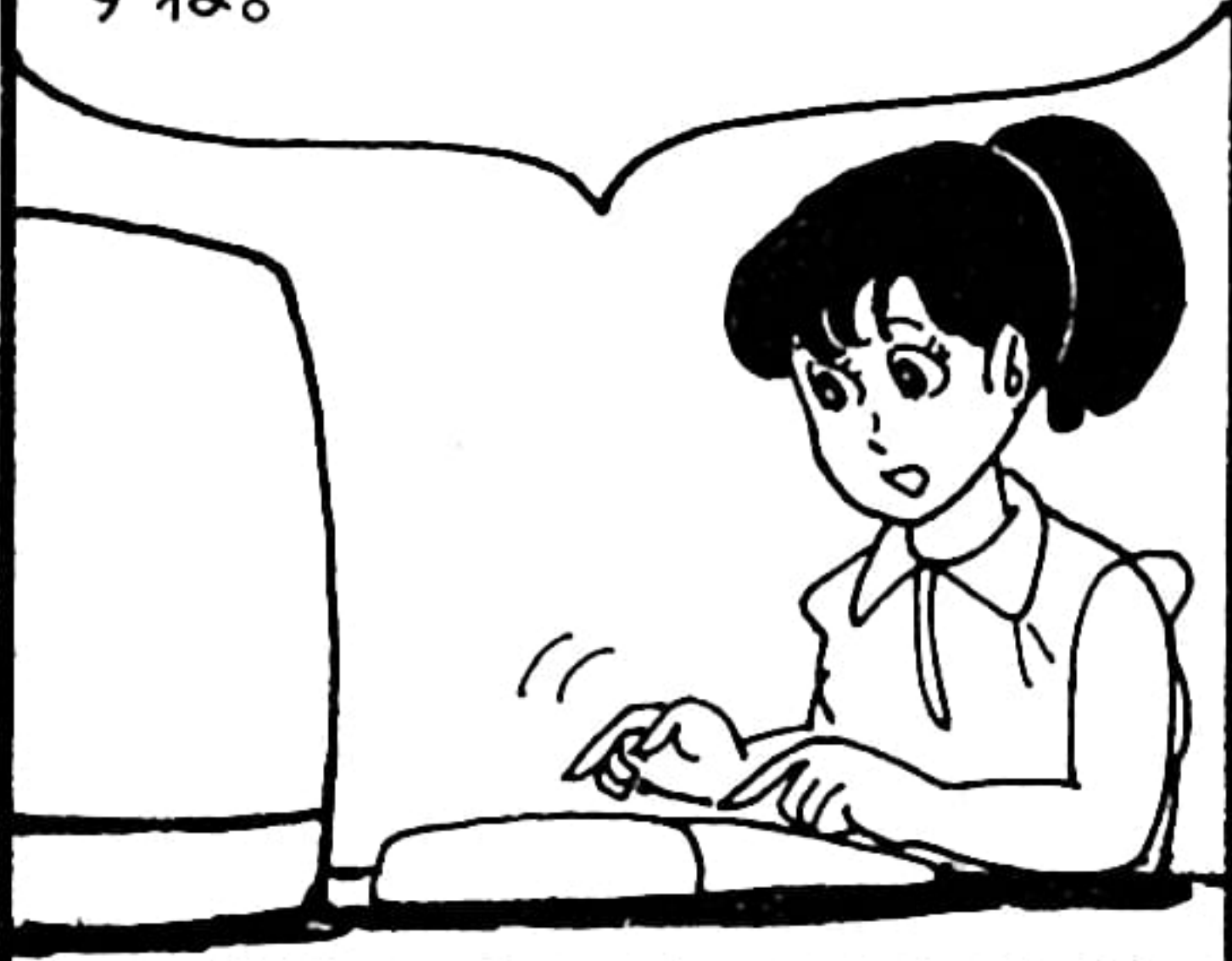
パソコンはいろんなものがあります。用途に応じて選べる時代になりました。

あなたが打ち込んでいたのはたぶんBASIC（ベーシック）というプログラム言語です。

LIST
10 DIM A(30)
20 INPUT "TOM
30 FOR I=1 TO N

ま、大変というよりどこから手をつけていいかわからないのが普通です。

でも、自分でプログラムして何かをやってみるとなるとこれが大変です。

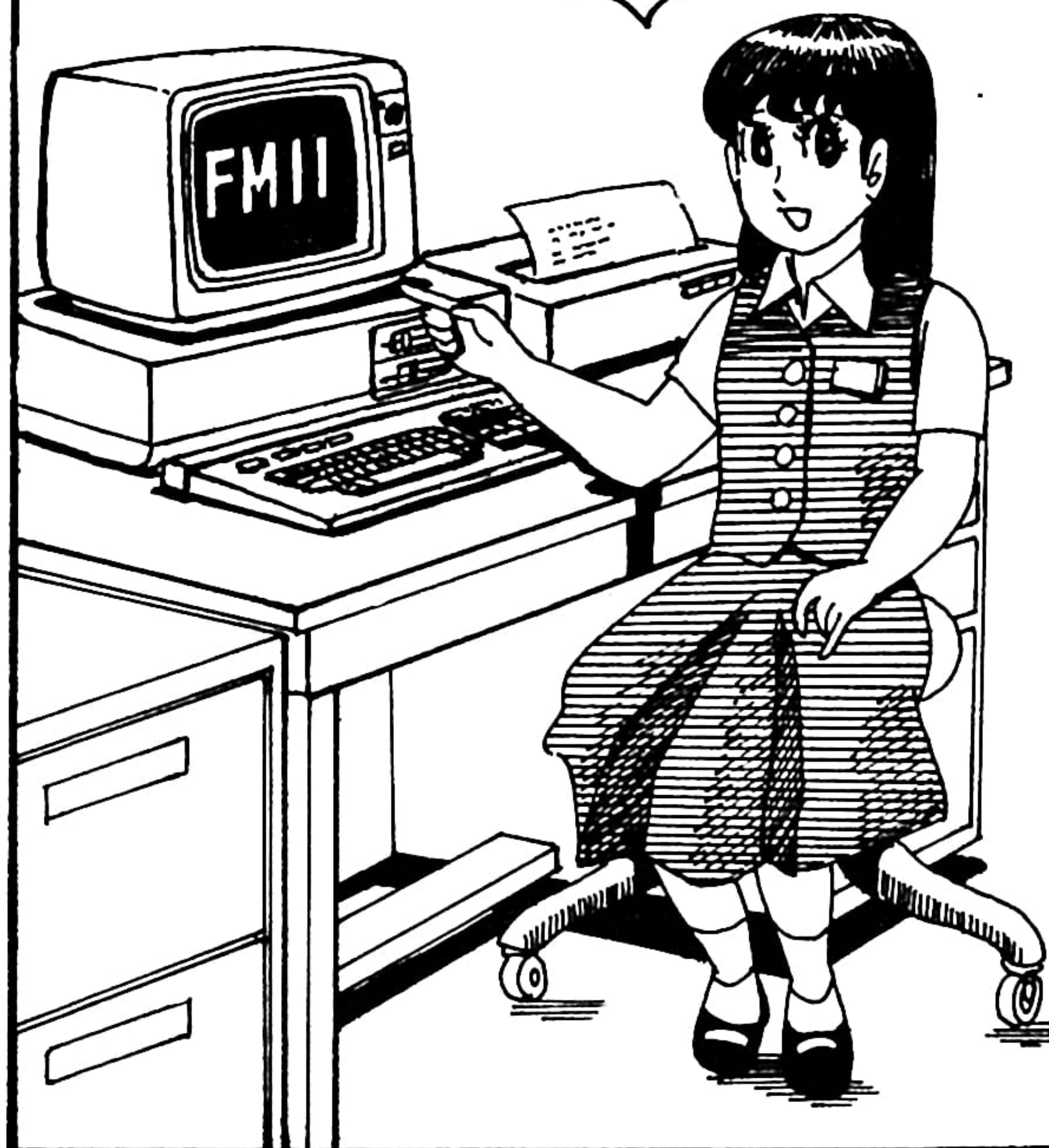
ここで使っているパソコンのBASICとあなたのものとは多少違うかも知れませんがコマンドのつながりがわかれば問題ないはずです。
では始めましょう。

パソコンの取扱説明書に載っている命令を実際どのように使ったらいいのか知りたいですね。

BASICはパソコン国へのパスポート。せっかくパソコンを持っているのならぜひ使えるようになってください。

BASIC インタプリタ空港

ワンボードマイコンは一部のマニアの間でしか普及しなかったのに、パソコンがここまで広がってきたのは、価格もさることながら、その使いやすさの違いです。ワンボードマイコンは機械語しか使えませんでしたが、パソコンでは人間の言葉に近いBASICなどの高級言語が使えるからです。

何年か前までは個人的に持てると言ったらワンボードマイコンだったのですが今では価格も安くなってパソコンが持てるようになりました。

FMII

ワンボードマイコンとパソコンの大きな違いは、そのメインメモリの容量の大小と言えます。

64Kバイトメモリ

RAM 256バイト

ROM 2Kバイトメモリ

ワンボードマイコン

パソコンでは、10進数、アルファベット、カタカナ、漢字などが使えます。そしてBASICなどの高級言語も使えるというわけです。

たとえばワンボードマイコンでは扱えるのは16進数だけで0～9、A～Fしか使えませんでした。

パソコンではメモリがたっぷり使えるので、ワンボードマイコンではできなかったことができるのです。

CPU

うーん、ぎっしりつまってる。

メモリ

0000

FFFF

モニタプログラム	QSプログラム		高級言語翻訳用プログラム		高級言語プログラム		機械語プログラム		データ、変数、スタック

たとえばメモリをこのように使うことができます。（メモリマップ）

FORTRANはIBMから発表された世界初の高級言語処理プログラムです。

コンパイラを使う代表的な言語はFORTRAN（フォートラン）です。

FORTRANプログラム言語の例

```
* EX TASHISA
INTEGER
REAL H;
S=0.0
READ(5,*)
```

FORTRAN 77

これがソフト・ウェアちゅうもんらしい。

高級言語はそれだけでは機械にわかりませんから、機械にわかる言葉に翻訳するプログラムが必要で、その形式によってコンパイラとインタプリタに分けられます。

コンパイラもインタプリタもその役目はプログラムの翻訳・通訳です。

たとえばFORTRAN で書いたプログラムを実行するためにはFOR-TRAN のコンパイラを使って全部機械語に直す必要があります。ただ、

コンパイル作業にはたくさんのメモリが必要なためディスクのついてないパソコンではFORTRANは走らないと思った方がいいでしょう。

翻訳したものを別々のメモリに入れる

0000

FFFF

OS（オペレーティングシステム）

コンパイラ（プログラム）

高級言語（プログラム）

機械語（プログラム）

インタプリタを使う言語ではインタプリタがプログラムの命令文をひとつずつとり出して翻訳実行するため機械語プログラム用のメモリ領域が不要です。

0000

インタプリタ（プログラム）

高級言語（プログラム）

これに対してBASICで書いたプログラムの実行はプログラムのインプット完了と同時にRUN RETURN の手軽さで行なえます。

BASICなどの高級言語を使うと、ハードウェアのこと、つまりCPUのポインタ、ワーキングレジスタ、メモリ領域などを考えなくてもプログラムすることができます。

B章でお会いしましょう。

ハード

```
10 INPUT "N=";N
20 A=0:B=0
30 A=A+B:B=B+1
40 IF B<N THEN GOTO30
50 PRINT "N=";USING"
60 LPRINT "A=";N
```

パソコンにインタプリタ形式のBASICが使われる理由はコンパイル作業のわずらわしさがないこと、メモリが少なくてすむこともあるのです。

たとえばBASICではコマンド、ステートメント、ファンクションの３種類に分類されるたくさんの記号（BASIC言語）があります。BASICでパソコンを動かそうと思ったら、このBASIC言語を覚える必要があるわけです。

コマンド（命令）	ステートメント（文）	ファンクション（関数）
AUTO	COLOR	ABS
CLOAD	DATA	ATN
CSAVE	DIM	COS
LIST	END	INT
MON	FOR	LEN
RETURN	NEXT	SIN
RUN	GOTO	USR
：	：	：

そのかわり、高級言語を使うためには、今度はその文法を勉強しなければなりません。

②ROM型パソコンとオールRAM型パソコン

同じBASICを使うパソコンでも電源を入れたらすぐ使えるものと、そうでないものがあります。

電源スイッチ

カチ

電源を入れたらすぐ使える方はBASICインタプリタがROM(ロム)というメモリに入っているものです。

0000　ROM　上位アドレス→

BASICインタプリタ

最初から入っている

一方、電源を入れただけでは使えずBASICインタプリタを入れる手間がかかるものはメインメモリ全体がRAM(ラム)になっているものです。

0000　RAM

最初はカラッポです。

RAMやROMについてはあとで詳しく説明しますからここでは気にしないでください。

RAM

ラムとロム

ROM

まずROM型パソコンを操作してみましょう。電源スイッチを入れるとすぐに何かメッセージが出てきます。たとえばPC-8001ではこのように表示されます。

NEC PC-8001 BASIC Ver 1.0
Copyright 1979 (C) by Micro so
OK

OKはよろしいという意味、つまりBASICインタプリタモードになっているということです。

OK

この状態はBASICのコマンドやプログラムの入力待ちです。前に何かプログラムを入れていてもそれは電源を入れた時、消(クリア)されています。

0000

BASICインタプリタ

ROM　RAM

少しややこしいかも知れませんがROM型の場合もプログラムを入れる部分はRAMメモリです。RAMは電源を切るとデータがメチャメチャになるため電源を入れた時にクリアしているのです。

0000　FFFF

ROM+RAM

メインメモリのハード構成

あとは、マニュアルに書いてあるとおりの文法でプログラムを入れてRUNすればいいのです。

RAMに入れて使うプログラムはテープなどで保管します。

次はオールRAM型パソコンです。これは電源を入れた時すべてのメモリエリアがクリアされそのままではどのプログラムも走りません。

そこでインタプリタを入力するための外部記憶装置が必要です。カセットテープやフロッピーディスクが使われます。

オールRAMにするためには0000～FFFF番地の中には何もプログラムしておけません。

どうしてインタプリタをロードするんだろう。

0000番地

RAM

そのためIPL（イニシャル・プログラム・ローダ）という機能が必要になります。

IPL

押釦スイッチの記号です。

0000番地

このIPL押釦を押すと、外部記憶装置からBASICインタプリタがメモリにロードされます。

カチッ

外部メモリ

BASIC インタプリタ

IPL

0000

RAM

ロードが完了したらIPLは見かけ上姿を消して、BASICモードになります。

IPL

バトン

BASIC インタプリタ

これでROM型パソコンと同じようにBASICが使えるのですが、なんでこんなめんどうなことをするのでしょうか。

OK

それはオールRAMの場合、ここにオペレーティング・システム・プログラム（OSといいます。）を入れることができるからです。

0000番地

OS

RAM

たとえば８ビットCPUのひとつである8080のOSとしてはディジタル・リサーチ社のCP／M(シーピーエム)があります。

CP/Mは８ビットCPUのOSの主流になりつつある。

このCP/Mを使うとBASICの他の言語のプログラムも使える利点があるのです。それはOSというのは言語開発にも使えるからです。話がむずかしくなりました。このCP/Mの使い方の例はあとの章ででてきます。

CP/Mはディジタル・リサーチ社の登録商標です。

CP/Mマシン（CP/Mの走るパソコン）ではIPLスイッチを入れると高速でフロッピーディスクのCP/Mプログラムがメモリにロードされます。

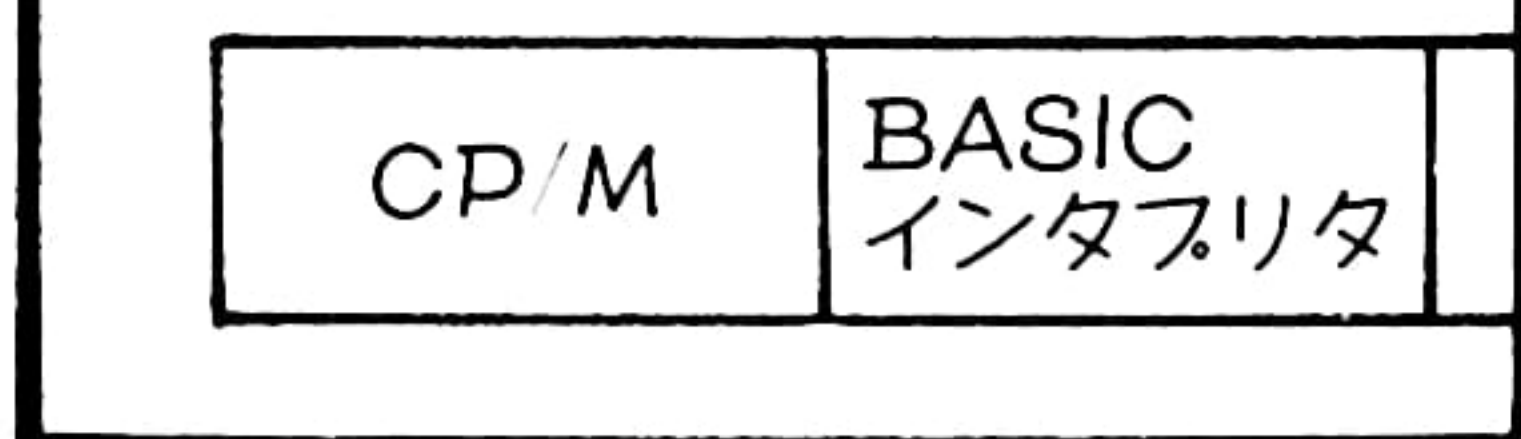
IPLスイッチON

カチッ

カチャカチャ

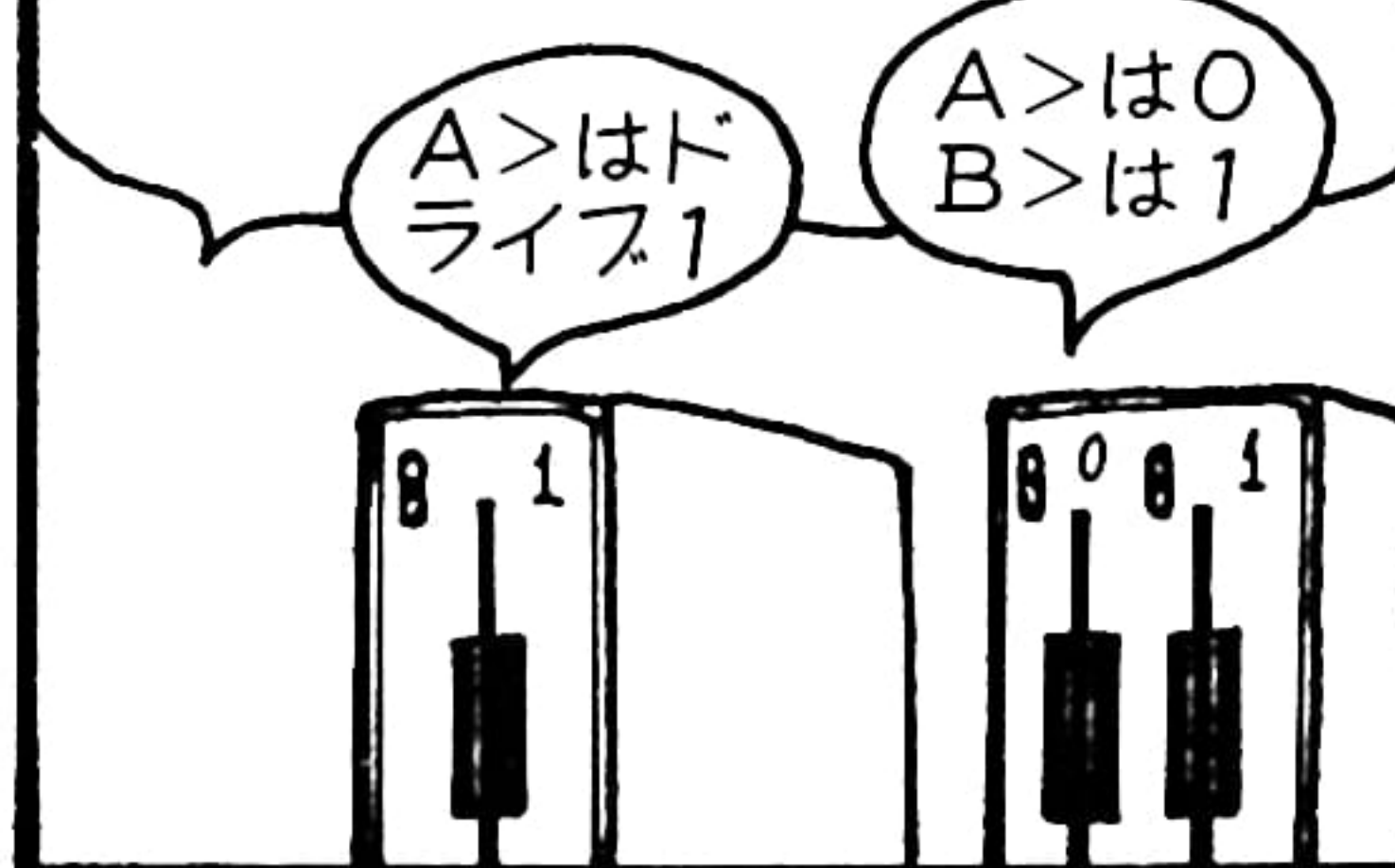
CP/Mが走る(なぜかこういう言い方をします。)パソコンは少なくとも1台のディスクドライブが必要です。CP/Mではカセットテープは使わないのです。

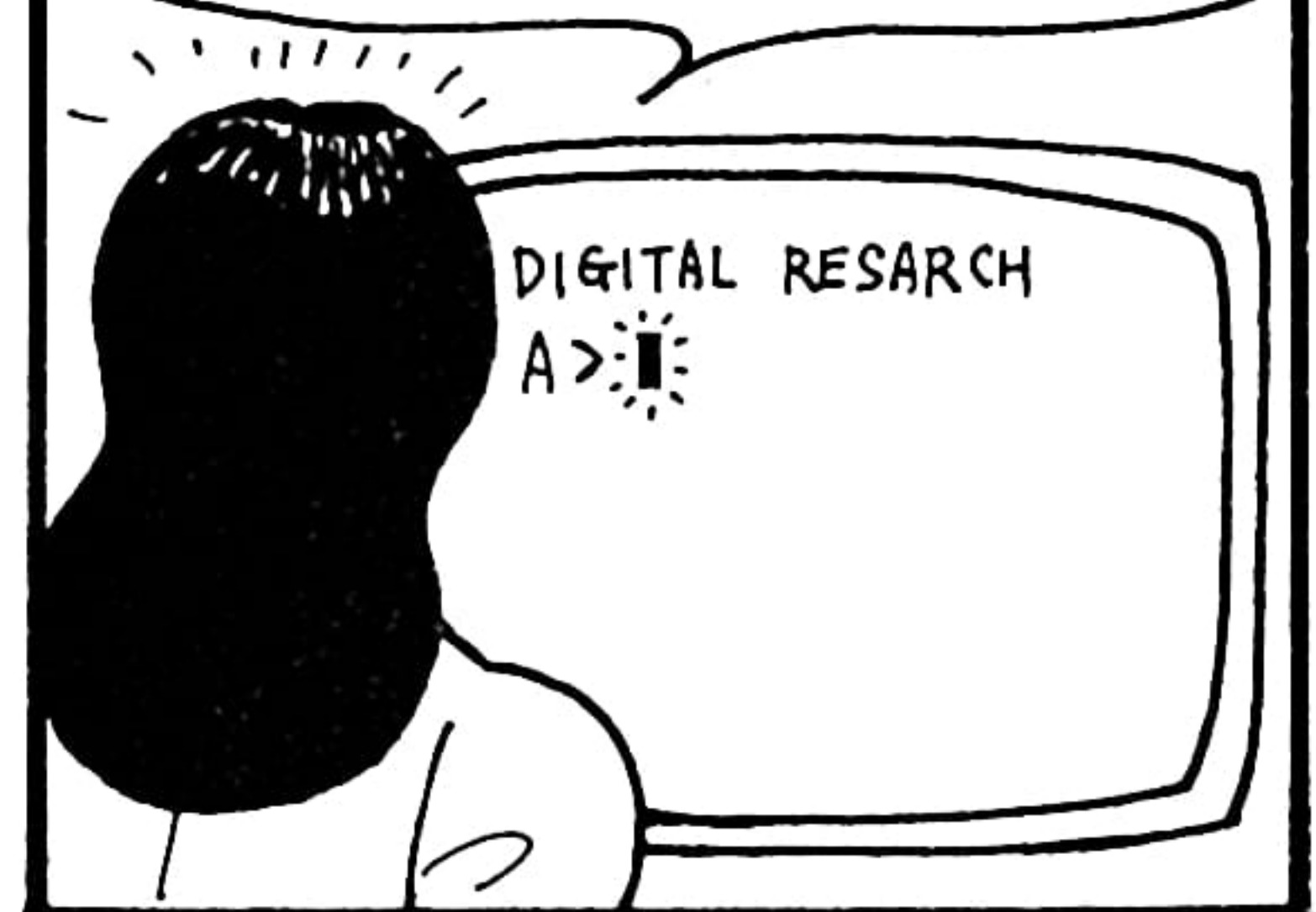
ディスクドライブ

CP／Mのモードになったら、こんどはキーボードからBASIC RETURN と入れてやるとフロッピーからBASICインタプリタがメモリにロードされます。

CP/M	BASIC インタプリタ

A>のAはドライブのNo.を示しています。日本のパソコンのドライブは0、1か1、2というNo.ですけどね。

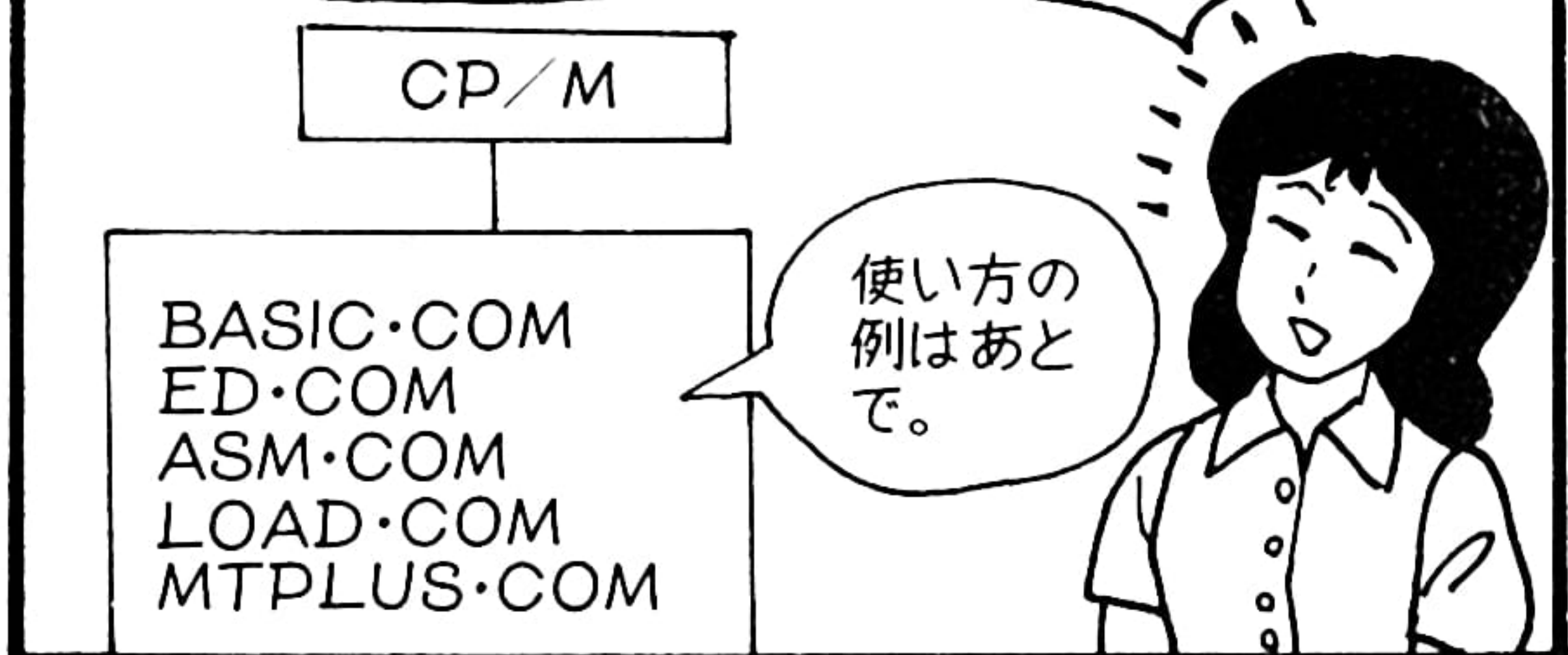
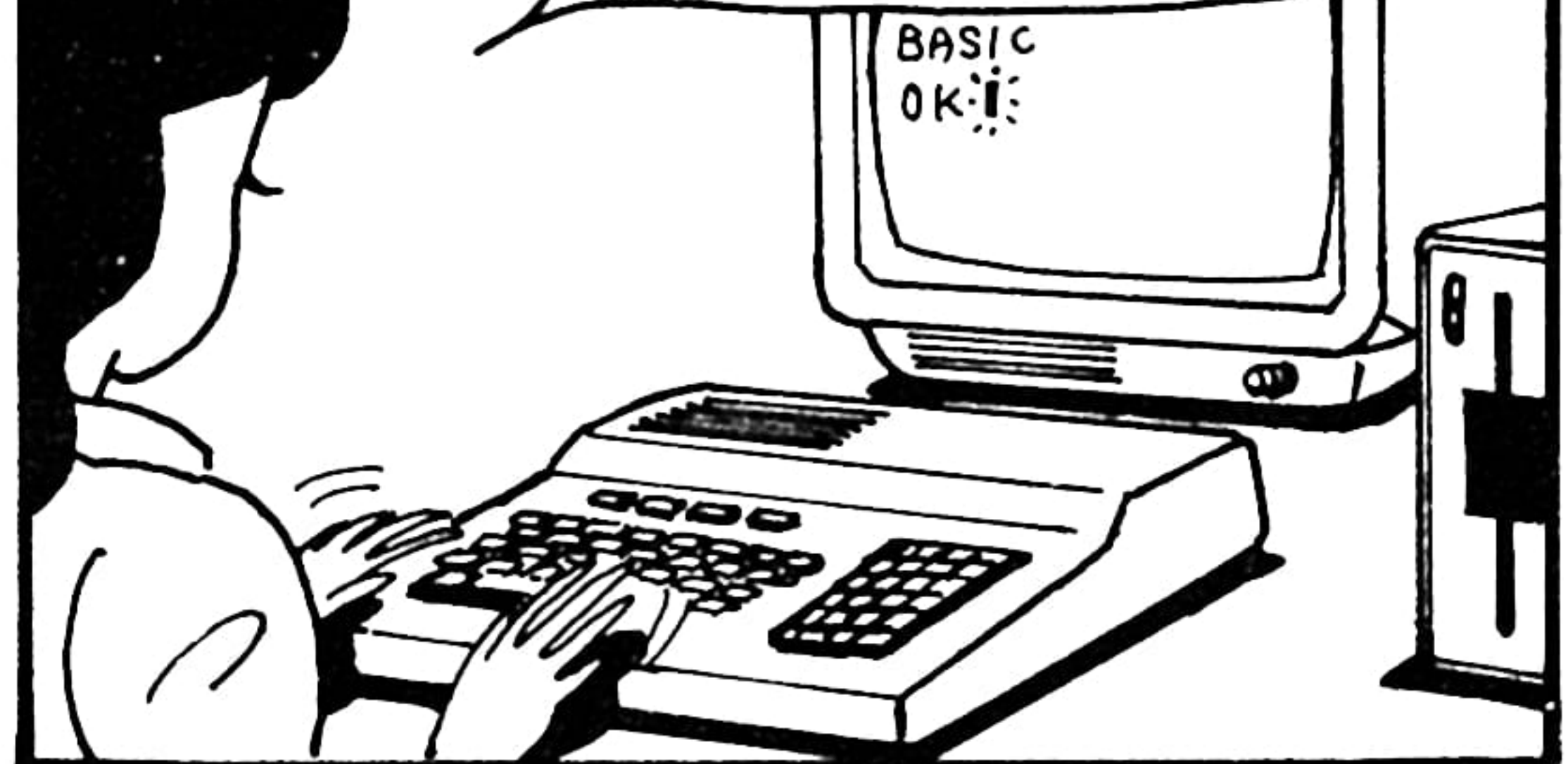
A>はドライブ1

A>は0 B>は1

BASICモードは＊印やOKですがCP／MではA>というような表示になります。

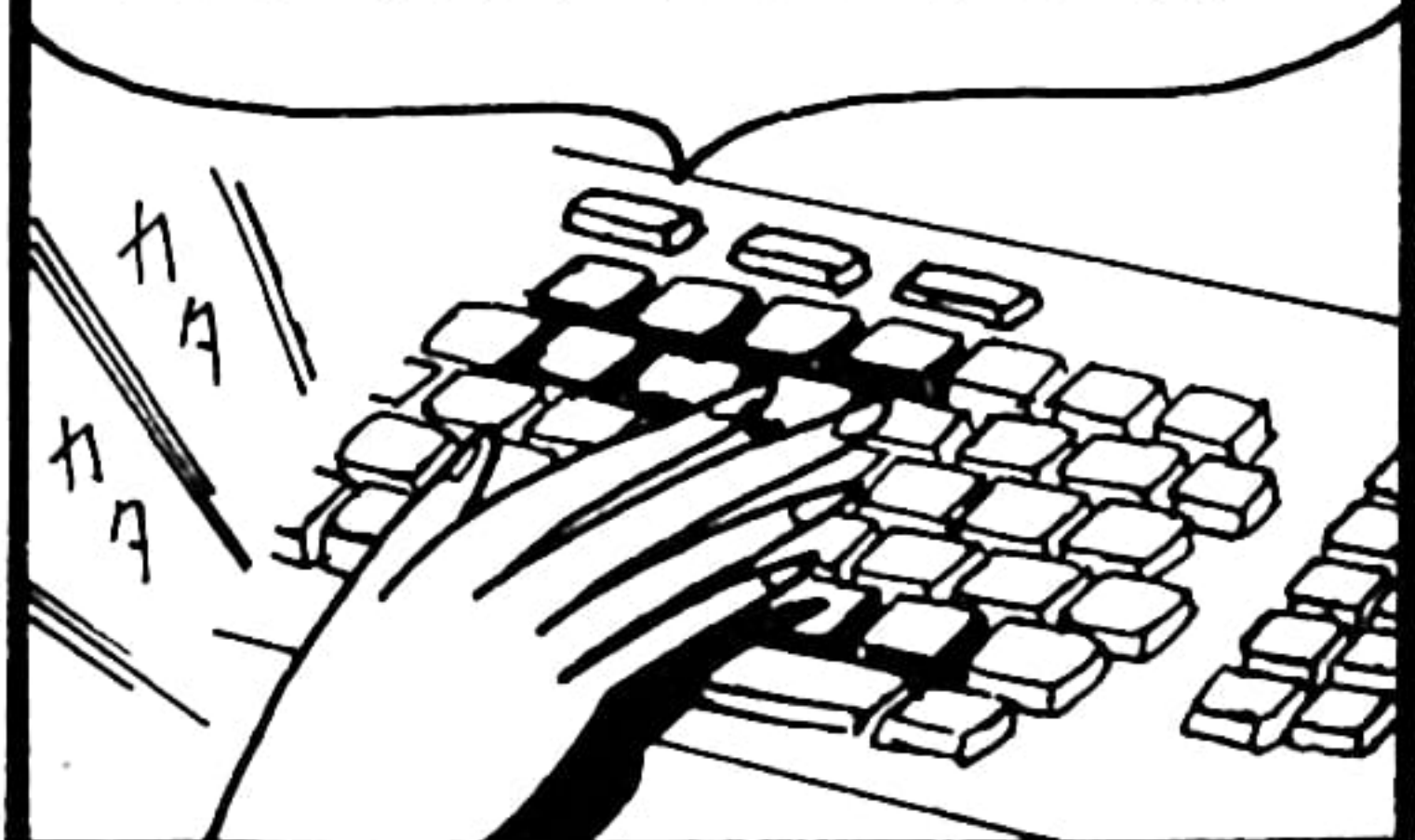
DIGITAL RESARCH
A>

今はBASICインタプリタを入れましたがこの他にもいろんなプログラムが使えるようになっています。

CP／M

BASIC・COM
ED・COM
ASM・COM
LOAD・COM
MTPLUS・COM

使い方の例はあとで。

ロードが完了するとBASICモードになりCP／Mは見かけ上消えてしまいます。

BASIC
OK

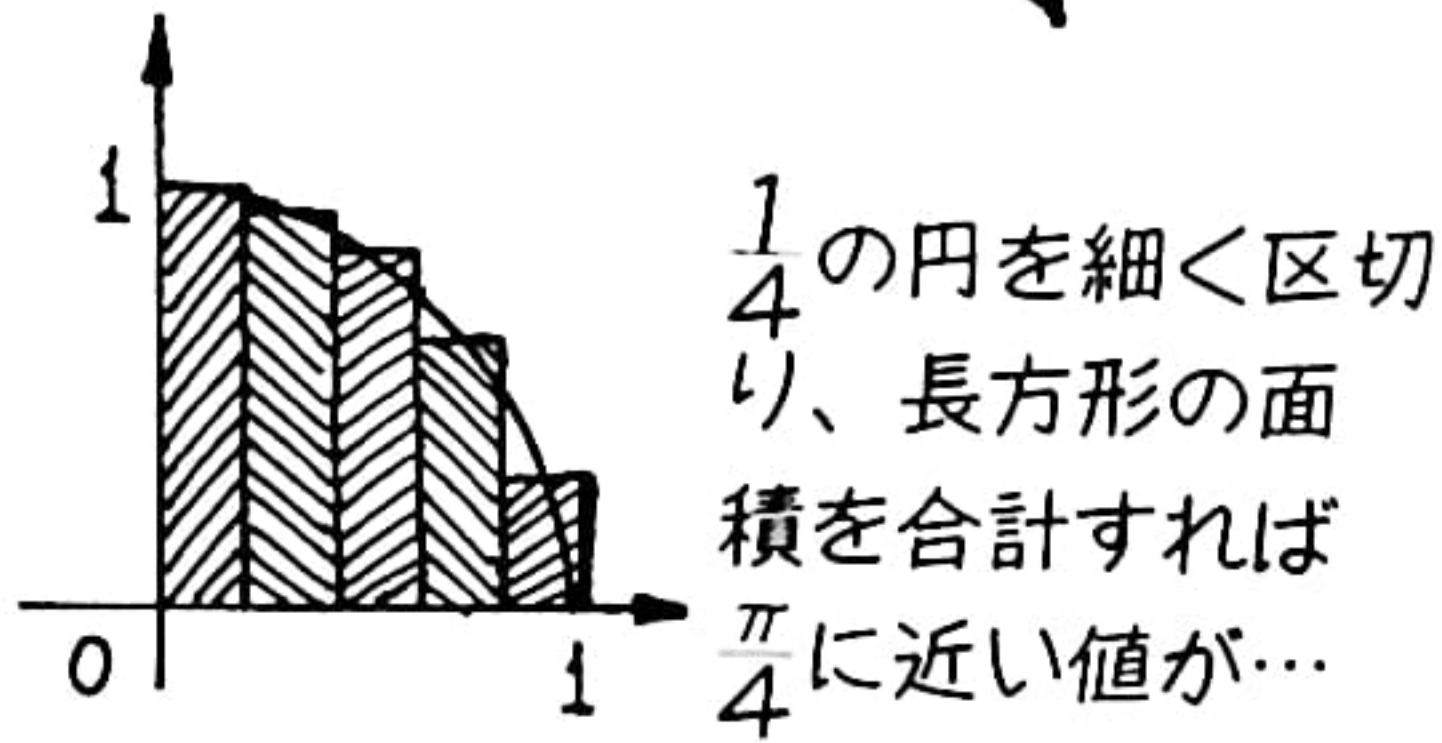
では、BASICでプログラムしてそれを走らせてみましょう。マニュアルを見てるだけではどう使っていいのかわからない記号がいっぱいありますから。

カタ カタ

BASIC言語だけで満足ならばROM型のパソコンが便利だし、成長に合わせて別の言語を使いたいのだったらオールRAM型パソコンを選ぶべきでしょう。

BASICだけかどうかそれが問題

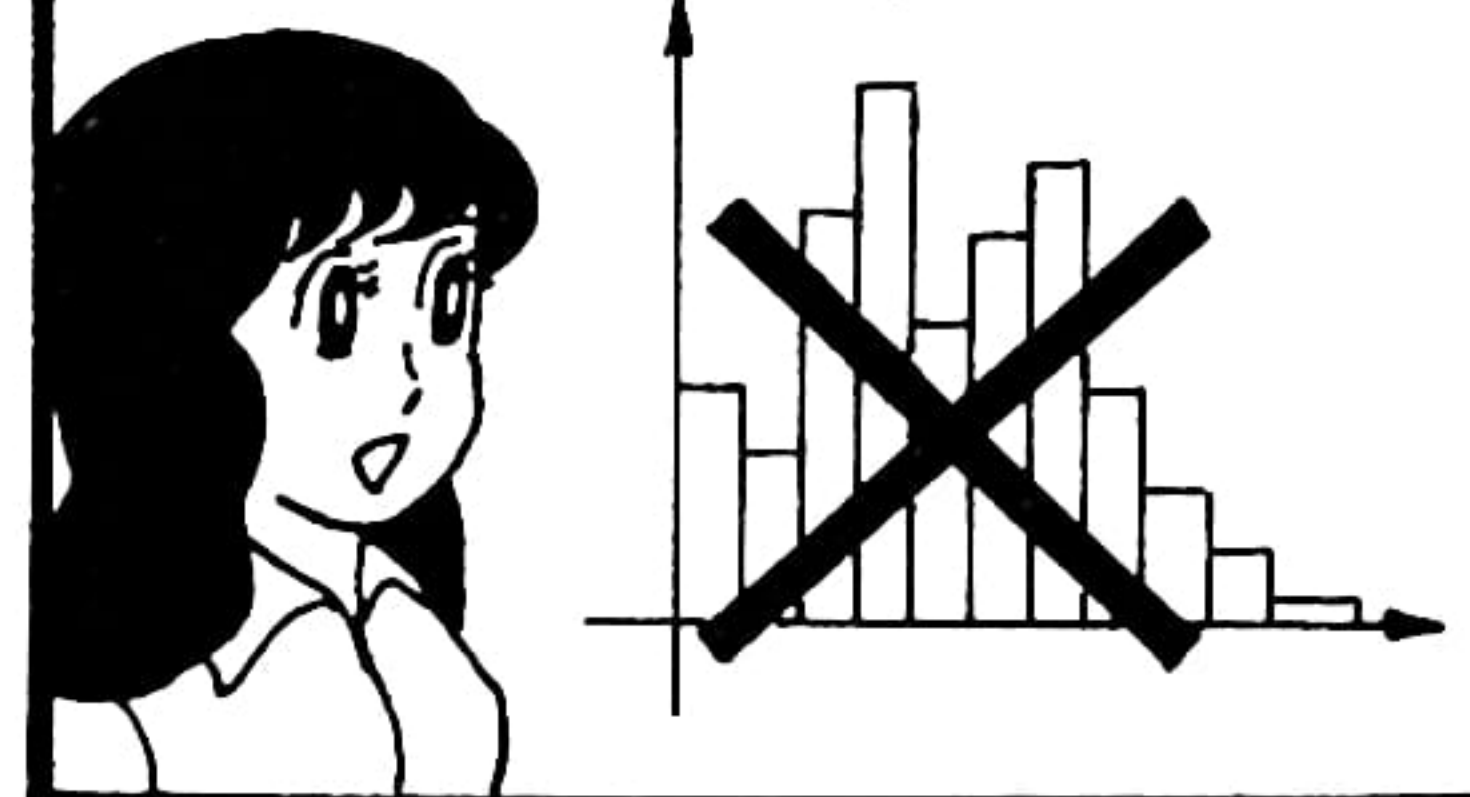
つまり、オールRAM型ではBASIC以外の言語やソフト開発（マシン語レベルのプログラムづくり）ができるということです。

次に繰返しルーチンのFOR～NEXT文の使い方の応用として半径１の円の面積からπ(3.14)を求めるプログラム。

$\frac{1}{4}$の円を細く区切り、長方形の面積を合計すれば$\frac{\pi}{4}$に近い値が…

用意したのはまず１からNまで１ずつ増加してそれを累計していくプログラム。

A＝１＋２＋………＋N

１から順に１ずつ増してNまでの累計Aを求めます。

その前にちょっとおことわりしておきますが、ここではグラフィック関係のプログラムは用意していません。

これらの例題でかなりのステートメントやファンクションを使いますのでこれまでチンプンカンプンだったマニュアルが読めるようになることはうけあいです。

マニュアル

最後にフロッピーを使って、データを保存するプログラムを用意しました。

プログラムのSAVEだけでなく配列などのデータも入れてみたら？

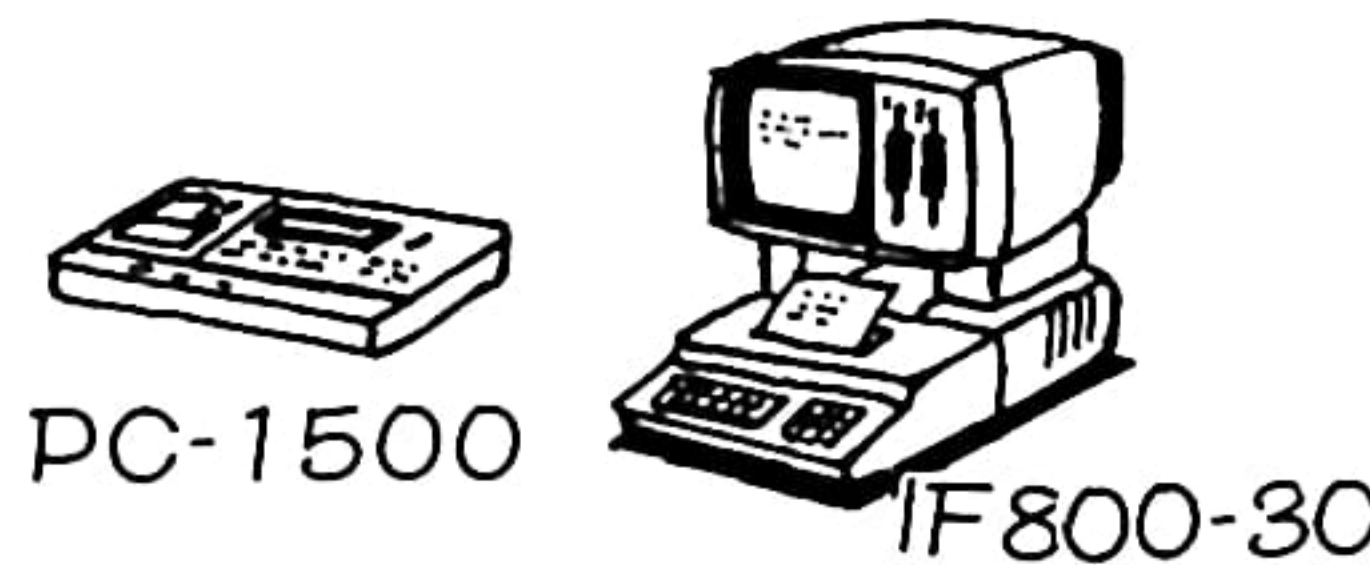
それから、表計算にとっても便利な配列を使ったプログラム。ビジネス向ソフトではなくてはならぬものです。

DIM A(3、20) とかで表現される配列と、実際の表のあやしい関係をスクープします。

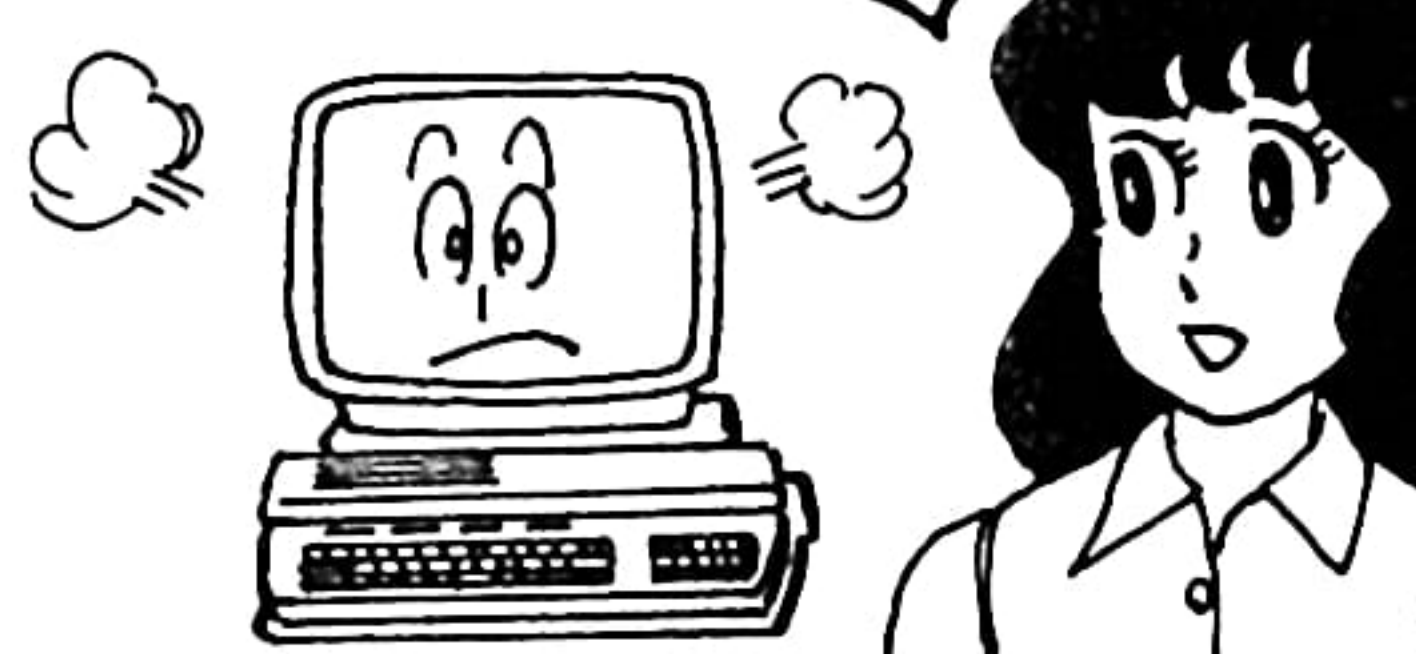
また、キーボードはメーカーによって違います。たとえば、データの区切りとして使うRETURNキーにしても、

OK AUTO
表示→ 10
キーイン→ AUTO RETURN

ここではたまたま手元にあったポケコンPC1500とパソコンIF800-30を使っていますが、もしFM-7やPC8001などを使ってもほとんど同様です。

PC-1500
IF800-30

ただBASIC言語と言っても現在のパソコン用に使われているものは各メーカーがいろいろ独自に開発し工夫している部分があって統一されていないということを知っておいてください。とくにRS-232Cという規格の通信ポートを使う場合などは本当にバラバラというところです。

そして何よりも実際にパソコンでキーをたたいてためしていただきたいと思います。IF800など100万円近くするし、持っている人は少ないと思います。ディスクが１台ついていてCP／Mが走るパソコンであればどの機種でも同じです。

これから見ていくサンプルプログラムも多少機種による違いが出てくるかも知れませんが、そこは大きな心で読み進んでいただきたいと思います。

あるメーカーではENTERまたあるメーカーでは↵印のキーを使っています。

OK AUTO
10
AUTO ↵

③ループを使って累加計算してみよう

1＋2＋3＋………＋N

同じところをグルグル回すことをループさせるといいます。ここではループのテクニックを覚えてください。

ミス・メモリ（おおぜいいます）

メモリ

変数名とその中身のデータ

データ

変数名

変数Aに累計の値を入れようと思ってこうしています。

A←0
A←0＋1
A←1＋2
A←3＋3
⋮

カウント

次に、変数Aをクリアします。Aには何が入っているかわからないので必ずクリアする必要があります。

A＝0

Aにゼロを入れる

0．0

このゼッケンはつけ換え自由

フローチャートを書いてみます。まずNをインプットします。

START

INPUT N

これは左辺と右辺が等しいということではなくて右辺を計算して左辺の変数にしまう、ということを表現する〝文〟なのです。

結果を入れる場所　A ⇦ A＋B 計算の内容

初めてプログラムを見る人がつまずくのがこの計算式です。右辺と左辺がどうして＝なのかわからないのです。

A＝A＋B
B＝B＋1

ループさせるためにもうひとつ変数Bを使い、これをカウンタにします。

B＝0（Bのクリア）
A＝A＋B（累計）
B＝B＋1（カウント）

この基本がわかっていると、BASICの文法の理解も早くなります。

A＋B＝A

この式は間違いであることがわかります。

このように同じ変数の中に計算結果を入れるようにするとメモリが節約できるのです。

この場合はA＋Bを計算して結果をAに入れるということです。もちろん計算後はAの前の内容は消されてしまいます。

むずかしい？

B＝Nになるまでここでぐるぐるとループを回っています。

ループ

A＝A+B
B＝B+1

B:N

B<N

1
2
3

3

たとえばN＝3とすると、Bの内容はこのように変化します。

(B)
0 ←0 --- Bのクリア

(B) (B)
1 ← 0 +1…1回目
2 ← 1 +1…2回目
3 ← 2 +1…3回目

次のB＝B＋1は1からNまでといった回数のカウントに使われます。(B＝B－1でもやはりカウントできるのです。)

1 2 3 4 5 6 … N

このフローチャートをまとめてみます。

これで求められたA、そしてインプットのNをプリントしてプログラムを完了します。

画面への表示

プリント

B＝Nの時にこのループからはずれるようにします。

◇の中はBはNよりも…である、を表わしています。

IF B>＝N THEN

B:N

B≧N

BはNよりも大きいか等しい

0→A
A+B→A
という具合に書くこともあります。

この中の数式を
A＝0
A＝A+B
としないで

プログラムを用紙に記述していくことをコーディングするといいます。

0→A
0→B

A+B→A
B+1→B

B:N

この表現では、0をAに入れるということ、また、A＋Bの結果をAに入れるということを感じとることができます。

A＝1＋2＋3＋……Nを求めるフローチャート

START

INPUT
N

A＝0
B＝0

A＝A+B
B＝B+1

B<N

B:N

B≧N

DISPLAY
N＝
A＝

PRINT
N＝
A＝

STOP

ひとつの答えを得るためにも、なかなかめんどうな手順が必要です。

これらはBASICやその他の高級言語の文法、つまり約束ごとなのですから、その原理さえわかれば、あとの数式表現でもいいと思うのです。

A=0
B=0
A=A+B
B=B+1

フローチャートのこのような表現から数式に直して転記する方がわずらわしい気がします。

0→A
0→B

A+B→A
B+1→B

でも、私は思うのですが、あとでプログラムを書く時はA＝0、A＝A+Bのように書くわけですから、

A=0
B=0
A=A+B
B=B+1

```
INPUT "N=";N
A=0
B=0
A=A+B
B=B+1
IF B<N THEN GOTO
PRINT "N=";USING "####";N
PRINT "A=";A
LPRINT "N=";N
LPRINT "A=";A
END
```

省略できる

CRTに表示

表示の桁数を決める

プリンタに印字

このフローチャートからBASIC言語でコーディングしてみました。

高級言語を使ったとしても、プログラミングし、それを完成させるまではエラーとの戦いなのです。エラーを誘うものはさけるべきです。

```
10  INPUT "N=";N
20  A=0
30  B=0
40  A=A+B
50  B=B+1
60  IF B<N THEN 40
70  PRINT "N=";USING"####";N
80  PRINT "A=";A
90  LPRINT "N=";N
100 LPRINT "A=";A
110 END
```

番号は適当につければよいのです。

あとで直しやすいように10とびにつけることが多いです。

この１行をステートメント（文）と言いますが、各々のステートメントに若い順に番号をつけておきます。これがBASICの特徴のひとつです。

No.　ステートメント

INPUT "N=";N

あとは、BASICの走るパソコンにこのプログラムを入れて試してみるだけです。今手元にはPC-1500（シャープ）がありますので、これを使いましょう。

これでフローチャートとプログラムコーディングが終わりました。

MODEキーを押してPRO（プログラム）モードにします。AUTO機能がないので文番号も全部キーインする必要があります。下のリストは 10：INPUT…のように文番号のあとに：がありますがこれはキーインしたのではありません。また、

60：IF␣B<NTHEN␣40 は

60IFB<NTHEN40 ENTER とキーインしたのであり、

60：IF␣B<N␣THEN␣40 ENTER としたのではありません。キーインが完了したらもう一度MODEを押してRUN（実行モードにします。

このポケットコンピュータはROM型パソコンなので電源をONするとすぐに使えるBASICが動きます。

他のメーカーのハンドヘルドコンピュータもROM型です。

STATUS 1 ← 使用済バイト数確認のためのコマンド

ENTER 表示→121

```
 10:INPUT "N=";N
 20:A=0
 30:B=0
 40:A=A+B
 50:B=B+1
 60:IF B<NTHEN 40
 70:PRINT "N=";
    USING "####";N
 80:PRINT "A=";A
 90:LPRINT "N=";N
100:LPRINT "A=";A
110:STOP
```

LLISTとキーインするとプログラムのリストがプリントされます。

この1行は 10INPUT "N";N ENTER とキーインしました。

このポケコンの場合はRAMに入れたプログラムがバッテリーでバックアップされています。電源を切ってもプログラムが消えないのがいい点です。

DEG PRO

このプログラムをRUNさせます。まず、N=_が表示されました。

カーソル

N=_

RUN ENTER

10をインプットしてみましょう。10と押します。

N=10_

ENTERを押すと、演算が始まり、結果が出ます。

N=␣␣␣␣10

この表示はN=␣␣10と表示されているのです。これはプログラムのUSING "####"のためです。

この桁は符号用

####

␣␣10

数値をアウトプットする時、その形式を指定しておく必要があります。

表示

プリント

BASICでは、USINGを使い、" "の中の＃で形を指定します。
今のプログラムをUSING "###.#" としておくと、
N=␣10.0
とプリントされます。

␣は空白の意味です。

次に ENTER を押すと、A=␣␣45が表示されました。これが1+2+…+10の答です。

A=   45

さらに ENTER を押すと、N=10、A=45が印字されました。これもUSING "####" の形式です。

N= 10
A= 45

一応プログラムは思ったとおり動いたようです。でも大事なのは答えが合っているかどうかですね。

足し算をしてみましょう。

1+2+3+4+5+6+7+8+9+10=
55

答えは55です。…?

この答えはコンピュータの方が間違っているようです。プログラムのチェックをしてみる必要があります。

高級言語を使っても、プログラムの完成までには、いくつかのエラーを解消する必要があります。この例のように、文法以外のエラーの発見は、システムが大きくなるほど、むずかしくなります。

この場合は、2つの答えの差が10ですから、最後の10を加えずにループを抜けたのだと判断できます。

LOOP

60:IF B<N THEN40のステートメントを

B<N
B:N

60:IF B<N OR B=N THEN40としてみます。

B<N
B=N
B:N

修正前のリストです。

```
10:INPUT "N=";N
20:A=0
30:B=0
40:A=A+B
50:B=B+1
60:IF B<NTHEN 40
70:PRINT "N=";
   USING "####";N
80:PRINT "A=";A
90:LPRINT "N=";N
100:LPRINT "A=";A
110:STOP→END
```

OR B=N

修正後のリストです。

```
10:INPUT "N=";N
20:A=0
30:B=0
40:A=A+B
50:B=B+1
60:IF B<NOR B=N
   THEN 40
70:PRINT "N=";
   USING "####";N
80:PRINT "A=";A
90:LPRINT "N=";N
100:LPRINT "A=";A
110:END
```

N=10、20、30、40と入れてみました。N=50ではオーバーで、ERRORになってプリントしませんでした。

```
N=  10
A=  55
N=  20
A= 210
N=  30
A= 465
N=  40
A= 820
N= 50
 ERROR
```

プリントフォーマット（書式）の指定の仕方によっては、答が印字されない場合があります。プログラムする時、十分考慮しておく必要があります。

変更後のリストです。

```
 10:INPUT "N=";N
 20:A=0
 30:B=0
 40:A=A+B
 50:B=B+1
 60:IF B<NOR B=N
    THEN 40
 70:PRINT "N=";
    USING "#######
    ";N
 80:PRINT "A=";A
 90:LPRINT "N=";N
100:LPRINT "A=";A
110:END
```

これはN＝50ではAが4桁となるためです。そこで、USING"####"をUSING"#######"としてみました。これだと6桁までいけます。

符号　有効数字

######

100回のループで5秒かかっているようです。
これで任意のNの計算時間を予測できます。

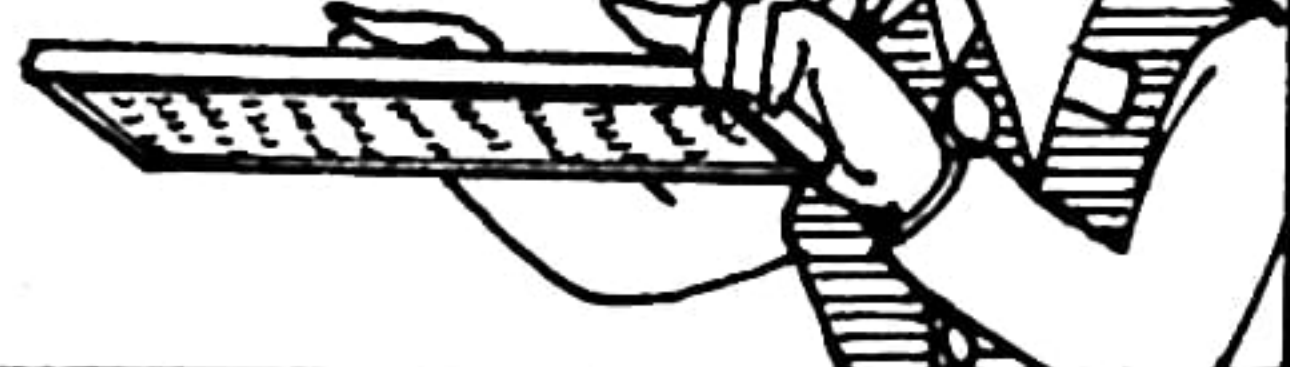
演算時間

```
N=      50  } 2秒
A=    1275
N=     100  } 5秒
A=    5050
N=     500  } 25秒
A=  125250
N=    1000  } 50秒
A=  500500
N=1500        1分15秒
A
          1125750
```

（N＝1500ではオーバーしたので Aの内容をマニュアルプリント）

N＝50を超えると、意外と計算時間がかかるので、その時間を測ってみました。
N＝1000になると50秒かかりました。

手計算で1＋2＋…999＋1000など、とても計算する気にはなれません。

1+2+3+4+5+6+7+8+9+10+11+12+14+15+16+17+18+19+20+21+22+23+24+25+26+27+28+29+30+31+32+33+34+35+36+37+38+39+40+41+42+43+44+45+46+47+48

でも、一度プログラムが完成すると事情が変わります。

N=1000では
R U N ENTER
1 0 0 0 ENTER
と押すだけです。

問題を処理するために、プログラムすることは、根気がいることであり、手計算した方が早いこともあります。

1＋2＋3…＋Nを求める公式がありましたね。コンピュータを使えて計算が苦にならなくなるとあまりものを考えなくなるかも知れませんね。

公式

$$1+2+3+\cdots+N = \frac{N(N+1)}{2}$$

N＝1000では

$$A=\frac{(N+1)}{2}=\frac{1000\times1001}{2}$$

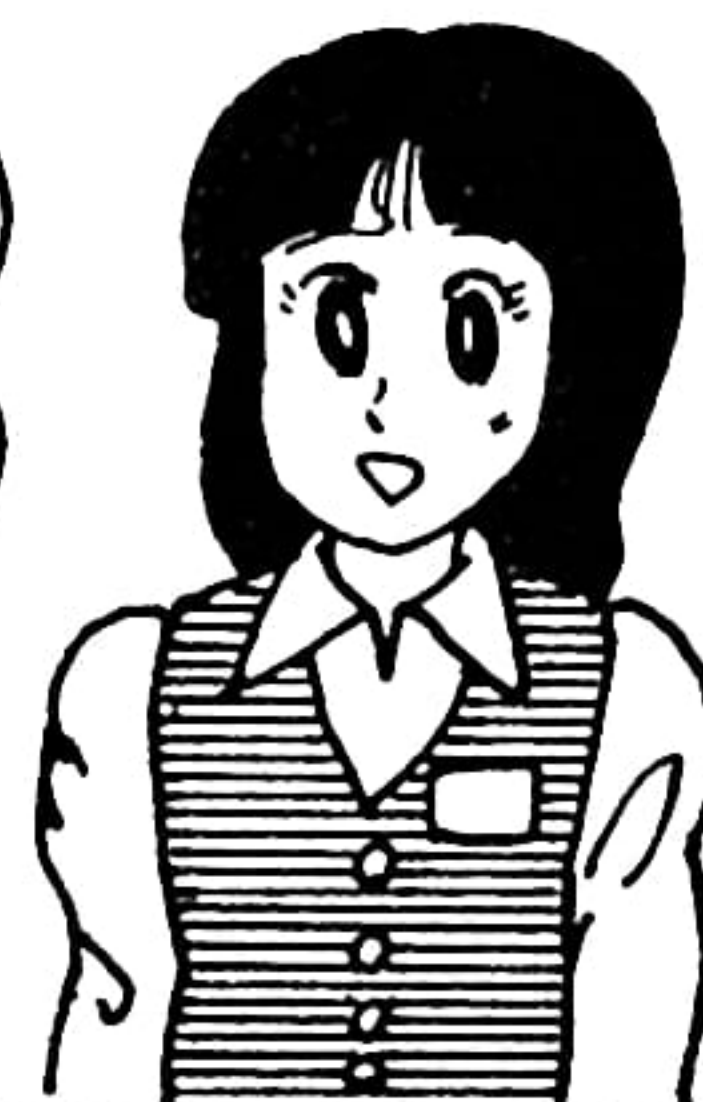
$$=500500$$

④FOR～NEXT文を使いこなそう

```
40 A=A+B
50 B=B+1
60 IFB<=NTH
EN40
```

はFOR～NEXT文を使うとこのように書けます。

```
FOR B=1 TO N
A=A+B
NEXT B
```

これを使えば、簡単にループルーチンを実現できます。

```
 10:INPUT "N=";N
 20:A=0
 30:FOR B=1TO N
 40:A=A+B
 50:NEXT B
 60:PRINT "N=";
    USING "#######
    ";N
 70:PRINT "A=";A
 80:LPRINT "N=";N
 90:LPRINT "A=";A
100:END
```

これを使って1+2+3+…+Nのプログラムを書き直してみました。

普通IとかJの変数を使う。

1からNまで1ずつBを変化させるという意味。

FOR B=1 TO N STEP1

この間の文の中にBがあれば1ずつ変化する。

NEXT B

STEP 1は省略できる。

同じ内容のプログラムでも、FOR～NEXT文を使った方が半分ぐらいに短縮されています。

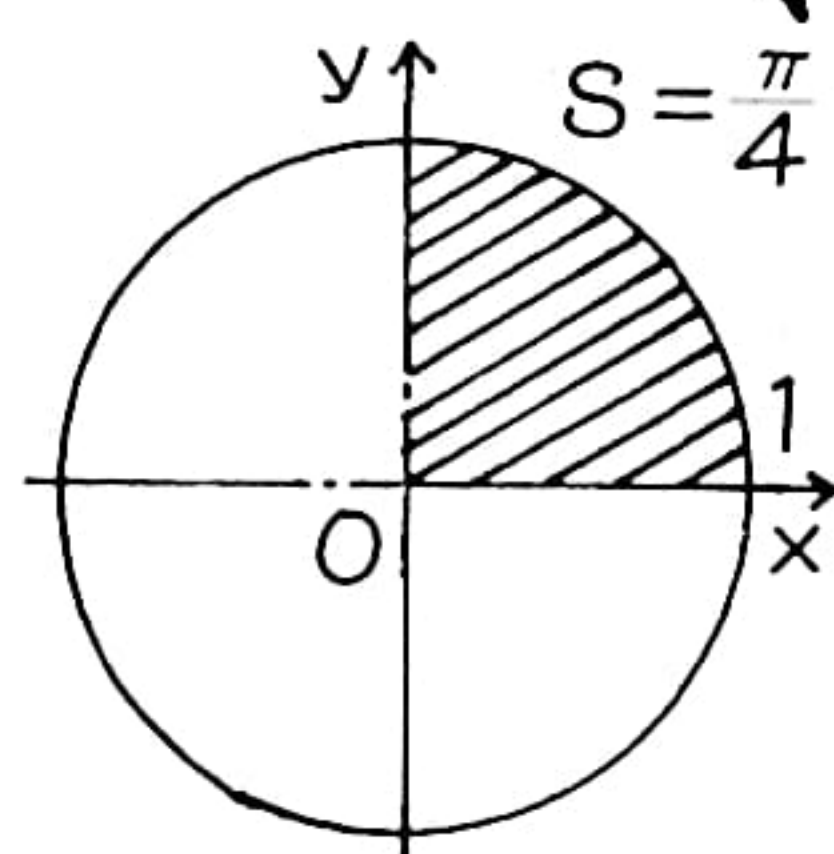
		演算時間
N=	50	1.6秒
A=	1275	
N=	100	2.9秒
A=	5050	
N=	500	13.2秒
A=	125250	
N=	1000	26.6秒
A=	500500	

さっそく、手元のポケコンでテストしました。

ついでに演算時間も測りました。

半径1の円の面積はπですね。
これの$\frac{1}{4}$の面積を求め、あとで4倍すればいいでしょう。

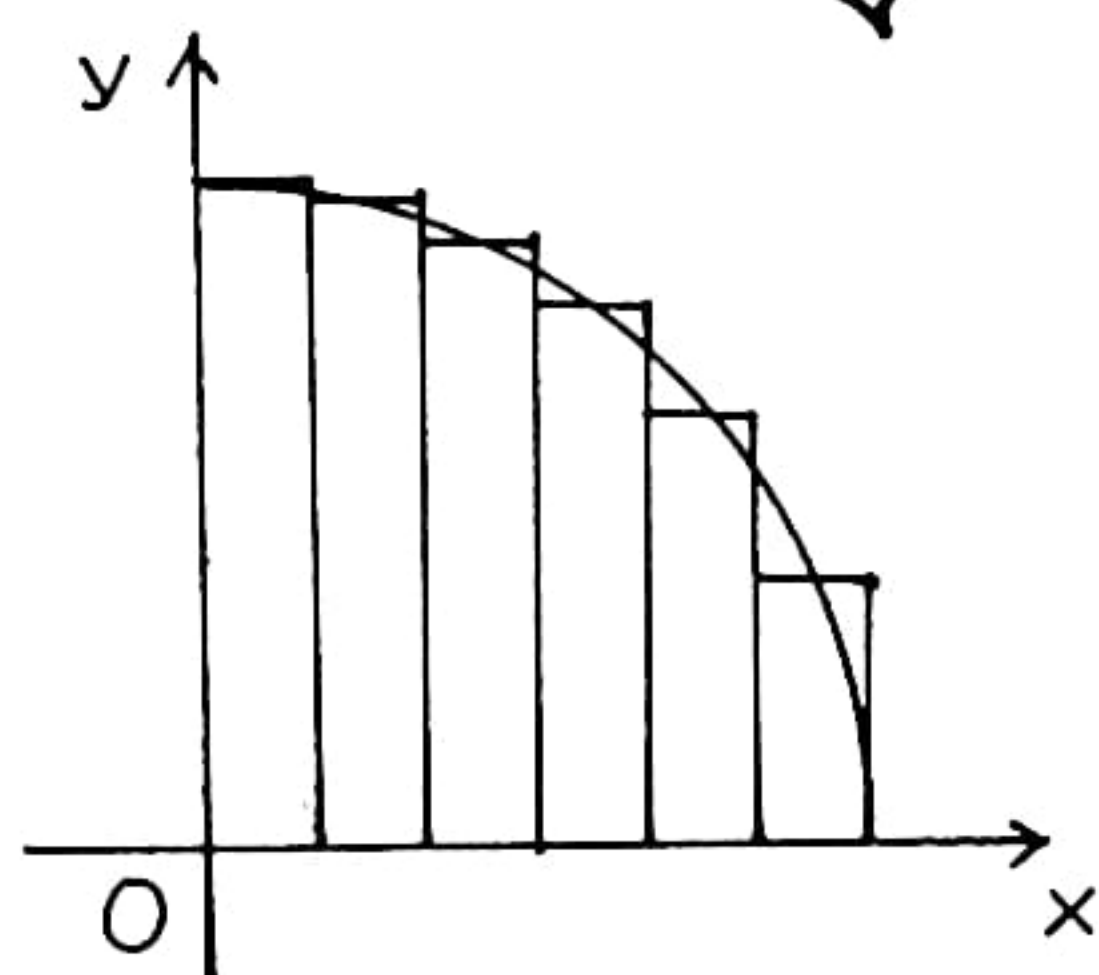
$S=\frac{\pi}{4}$

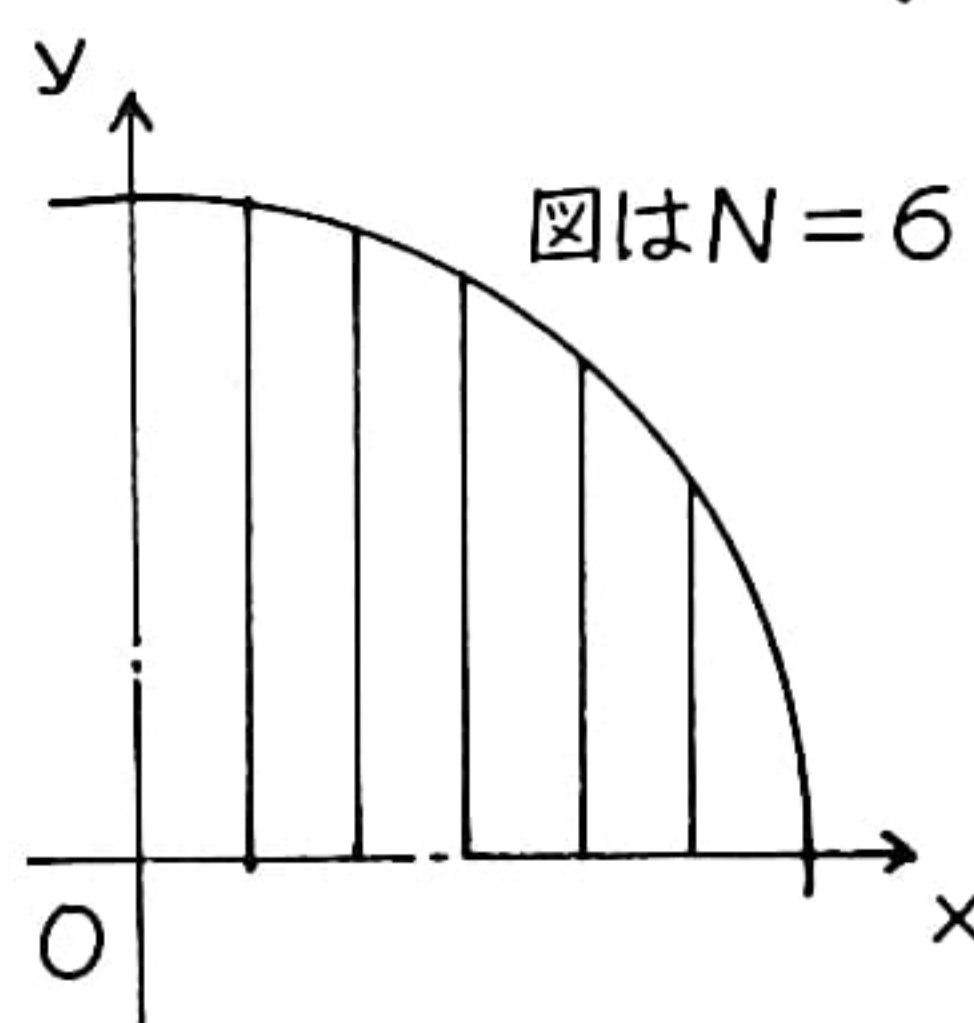
Y X O 1

ではここで、ひとつ別の問題をやってみましょう。円周率πを求めてみたいと思います。

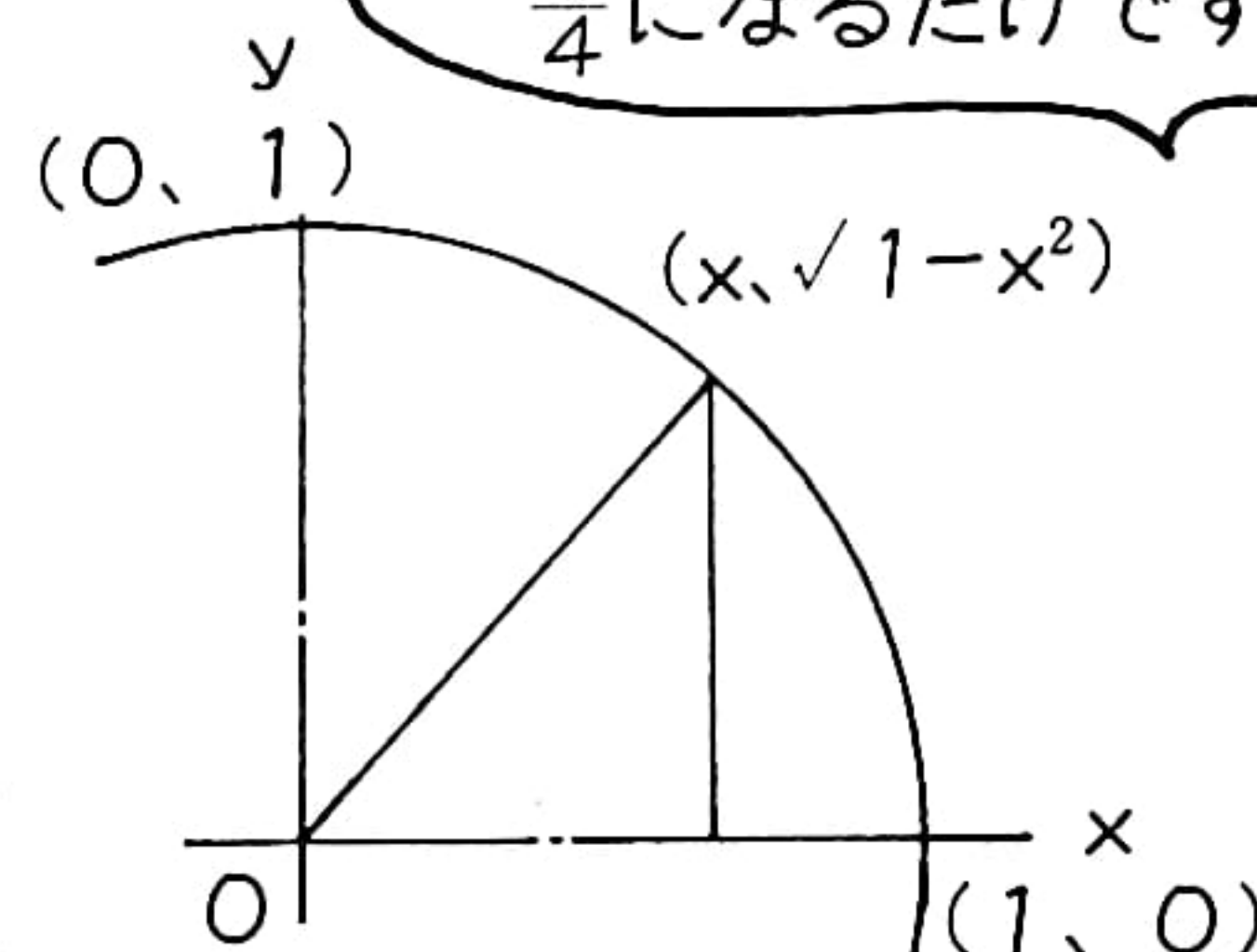
$\pi=3.14\cdots$

繰返し演算のある場合、プログラムの作り方によって処理時間が大きく開くことがあります。

このN分割されたそれぞれの長方形の面積を合計すると、その値は$\frac{\pi}{4}$の近似値になります。

y
O
x

コンピュータがあると、こんな求め方ができます。xをN分割します。

y
図はN＝6
O
x

πの値を使わずに、この面積を求めるのはむずかしいですね。積分しても$\frac{\pi}{4}$になるだけです。

y
(0、1)
$(x, \sqrt{1-x^2})$
O
x
(1、0)

プログラムするために、計算方法を決めなければなりません。まず初期値ですが、区分$\frac{1}{N}$のさらに$\frac{1}{2}$とします。あとは$\frac{1}{N}$ずつ増加させそれに対応したyを求めます。

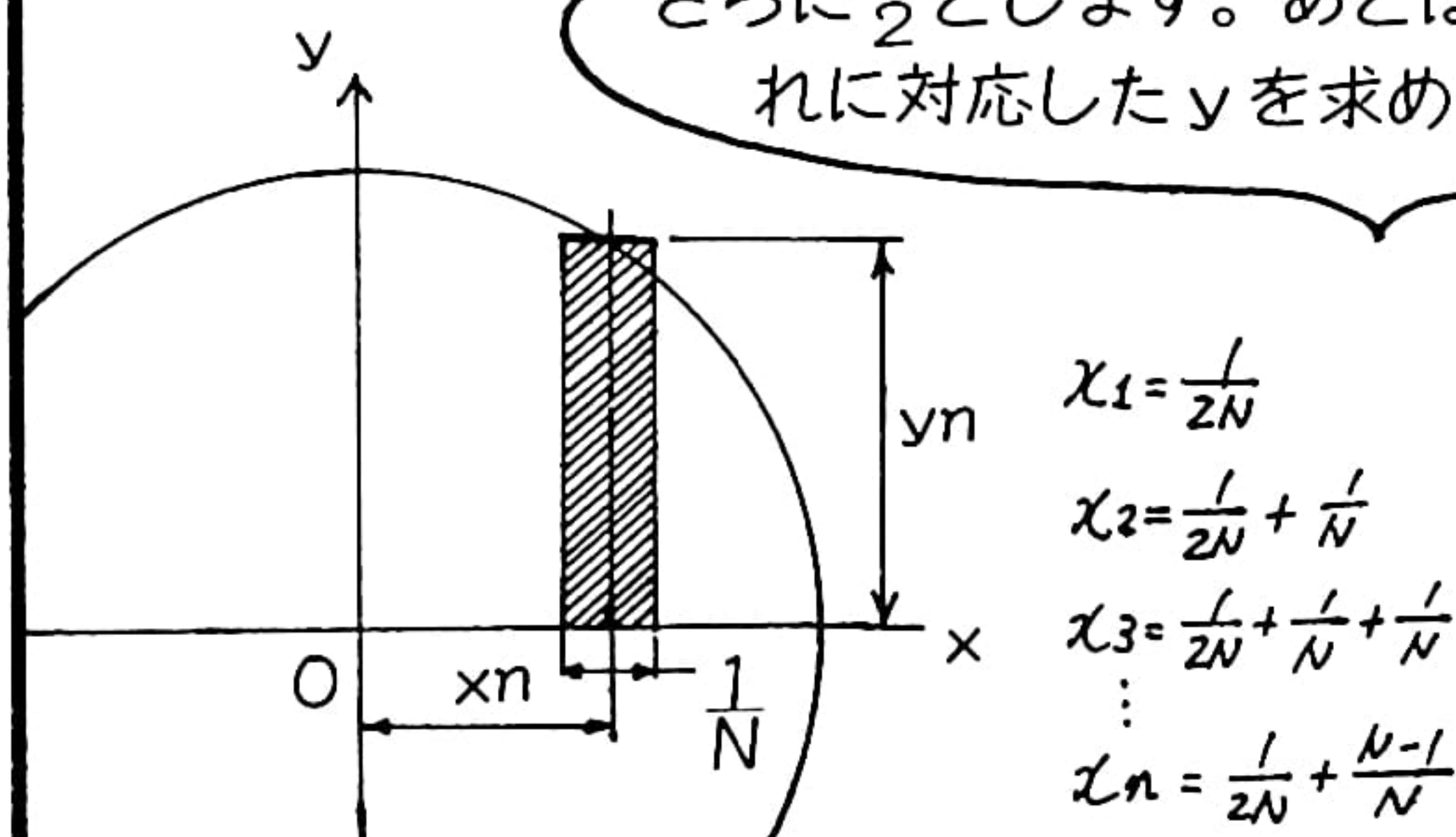
y
yn
O
xn
$\frac{1}{N}$
x

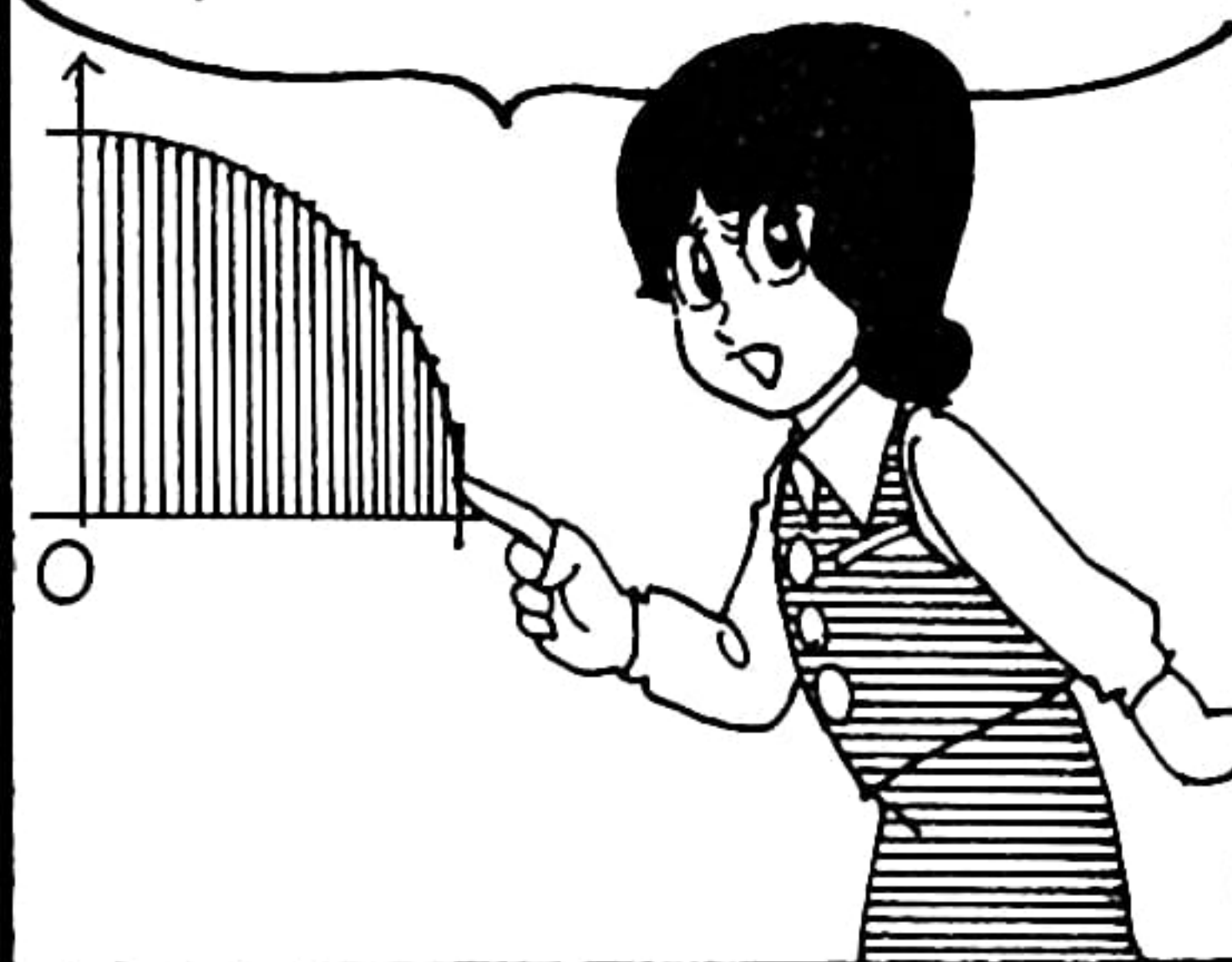
$x_1 = \frac{1}{2N}$

$x_2 = \frac{1}{2N} + \frac{1}{N}$

$x_3 = \frac{1}{2N} + \frac{1}{N} + \frac{1}{N}$

⋮

$x_n = \frac{1}{2N} + \frac{N-1}{N}$

この分割を十分大きくすると、かなりπに近い値が得られるはずです。

O

πの近似値をFOR～NEXTで

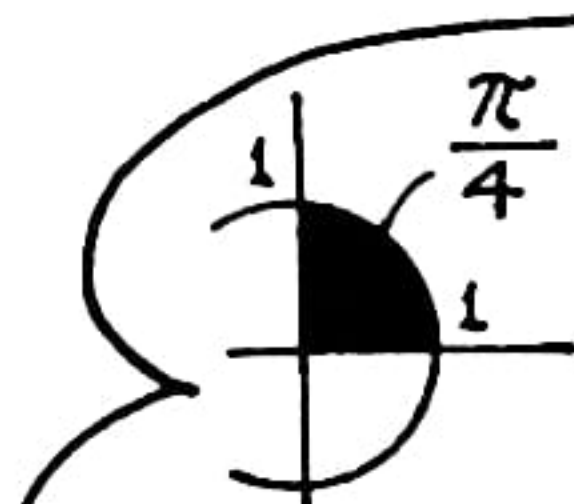
```
10  WAIT 0
20  INPUT "N=";N
30  T1=TIME
40  B=0:C=1/N/2:D=1/N
50  FOR I=1 TO N
60  A=D*√(1-C*C)
70  B=B+A
80  C=C+D
90  PRINT USING "######";I
100 NEXT I
110 B=4*B
120 T2=TIME
130 LPRINT "N=";USING "######";N
140 LPRINT "π=";USING "##.########";B
150 LPRINT "S.TIME=";USING "######.####";T1
160 LPRINT "E.TIME=";T2
170 LF1
180 GOTO 20
190 END
```

変数の意味

A…△S
B…S（文110でπになります）
C…x

TIMEはポケコンの現在時刻を得るための関数です。プリンタがついていない時は文130のLPRINTでエラーになります。

プログラムそのものよりも問題をコンピュータ向けに分析する能力を身につける方が実務的です。アセンブラ言語のようにプログラムそのものに時間をとられるようではコンピューターを使う意味が減ってしまいます。人間的発想でインプットした問題が解ける言語はこれからもますます必要になってきます。

$\frac{\pi}{4}$
1
1

長方形の面積は、区分$\frac{1}{N}$と高さ$yn = \sqrt{1-xn^2}$から求められます。

$\Delta S_n = \frac{1}{N} \cdot y_n$

$= \frac{1}{N} \cdot \sqrt{1 - x_n^2}$

$S = \Delta S_1 + \Delta S_2 + \Delta S_3 + \cdots + \Delta S_n \fallingdotseq \frac{\pi}{4}$

求められたSを4倍すれば、πの近似値が得られます。次のコマの文番号40から110までがπを求めるためのルーチンです。このプログラムはポケコンPC-1500のBASIC言語を使っています。

ためしにN=10としてやってみると π=3.1524…と出ましたので、分割を多くして、やってみました。

分割を多くすると、時間もかかります。N=10,000では、32分57秒かかっています。もちろんこれはPC-1500のスピードです。

Nの最大値32767までやってみました。

```
N=  5000
π= 3.141593628
N= 10000
π= 3.141593008
N= 15000
π= 3.141591457
N= 19000
π= 3.141591691
N= 20000
π= 3.141592793
N= 32767
π= 3.141593548
```

PC-1500のπの値は、3.141592654です。

これを見ると、信頼できるのは、3.14159までです。

細かく分割すればするだけ精度が上がるというものでもなさそうです。

BASICでの通常の有効桁は6桁、倍精度で12桁です。

この計算では、平方根も入っていますから、累積誤差も大きくなる可能性があります。

コンピュータの最小単位は0か1かの1ビット 小数点以下の計算はニガ手なのです。小数点以下のケタ数が多いと、累計した時も誤差が大きくなります。

2進		10進
0.1	……	0.5
0.01	……	0.25
0.001	……	0.125
0.0001	……	0.0625

技術計算で、繰返し計算をする時は、精度面で注意する必要がありますね。

B
0.00001
π
1 2 3 N
万回

FOR~NEXT構文は、使い慣れてくるととても便利です。同じ構文は文こそ違え、FORTRANその他のコンパイラ言語にもあります。

FOR
NEXT

⑤配列がわかればメモリもわかる

BASICにかぎらずマイコンがあつかうデータにはデータそのものと、そのデータを入れるところがあります。BASICではこのデータを入れておくところを文字で表わせました。

データ

データ

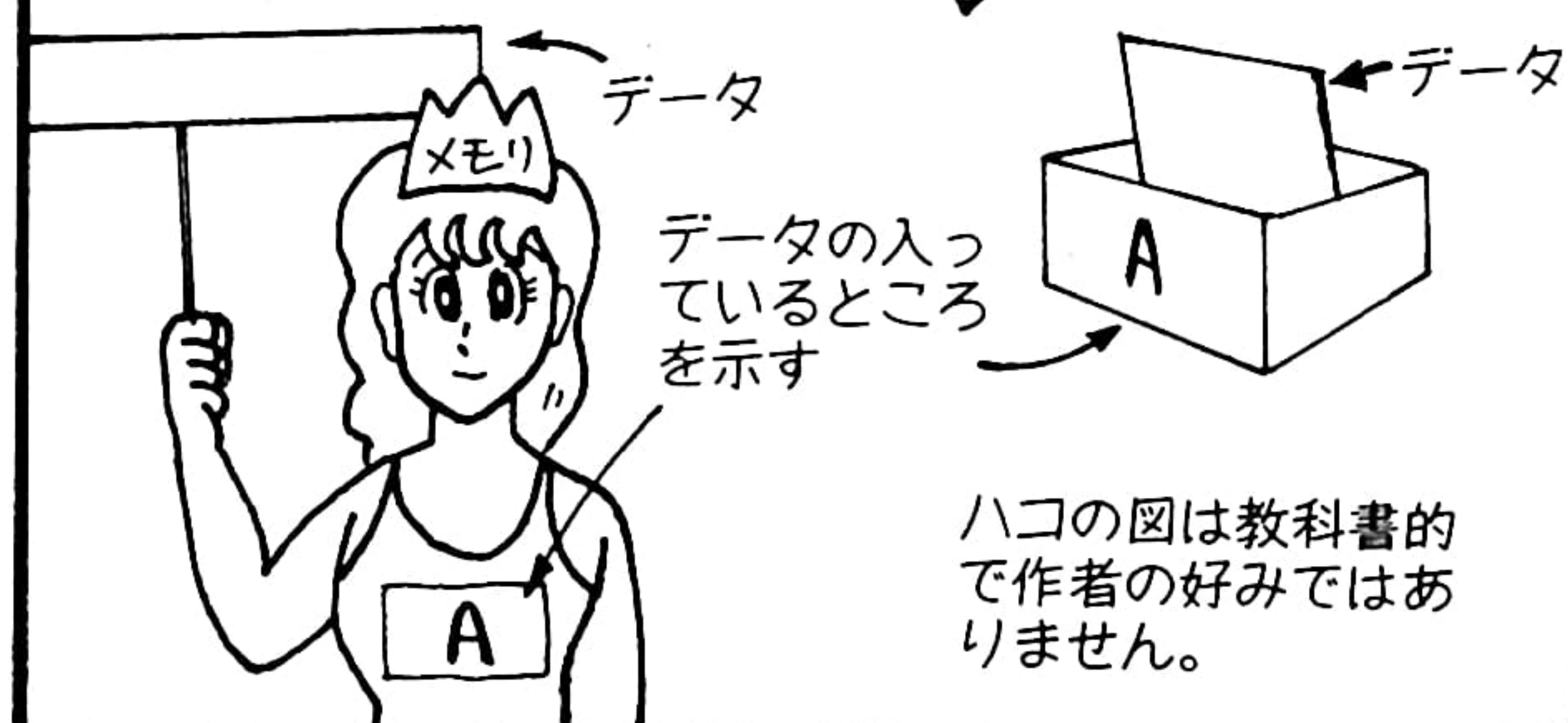
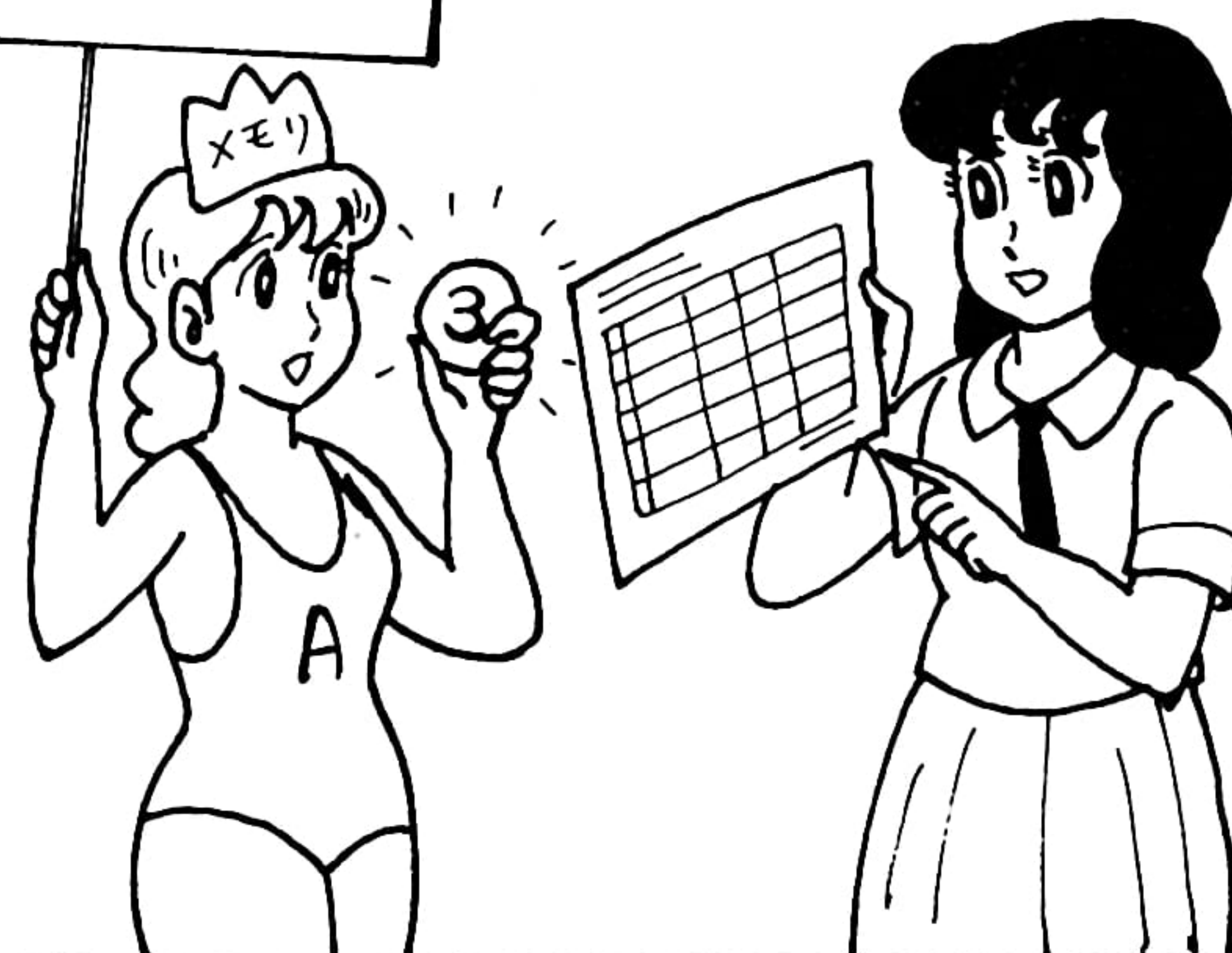
データの入っているところを示す

A

A

メモリ

メモリ

ハコの図は教科書的で作者の好みではありません。

このAやBは制限はあるものの自由につけてよくABC=DEFでもいいのです。このA、B、ABC、DEFといったものはプログラム実行時中身がコロコロ変わるわけで変数と呼ばれます。式の中に出て来てプログラムで変化しないものは定数と呼ばれてます。

A=B+ABC+DEF+2.217

変数

定数

たとえばプログラムでA=Bとすれば、プログラムをRUNさせてこの文が実行されると、Bの入れものに入っているデータと同じものをAの入れものに入れるという動作をしますね。

5.23

実行前のデータは消える

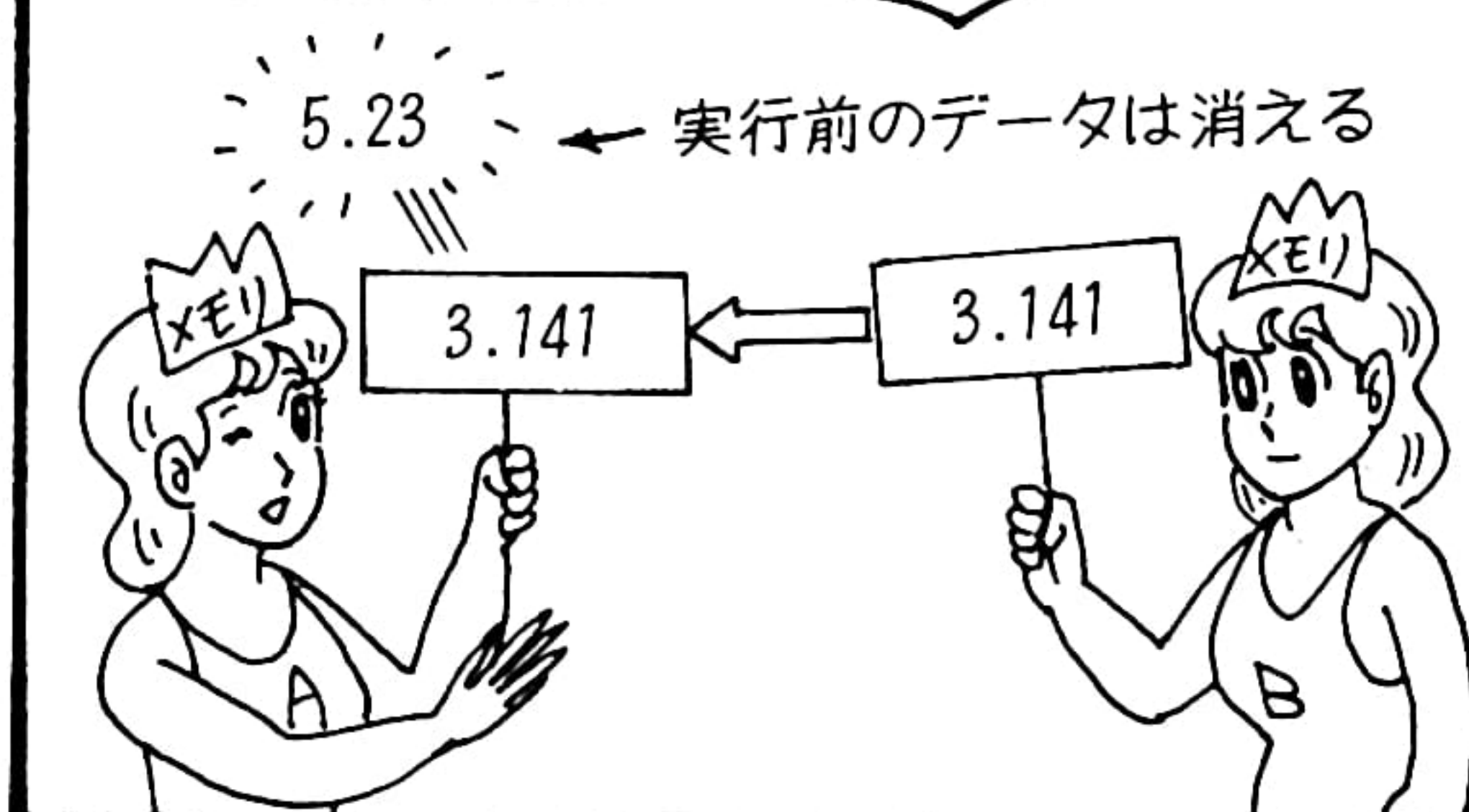
3.141

3.141

メモリ

メモリ

A

B

一方先生は生徒別の点数表が必要であり、
鈴木君の国語は何点、数学は何点……
佐藤君の国語は何点、数学は何点……
山田さんの国語は何点、数学は何点……
田中さんの国語は何点、数学は何点……
というような内容を表にします。これは二次元のデータになります。

科目＼氏名	鈴木	佐藤	山田	田中
国語				
数学				
地理				
物理				

ところで私たちが日常取り扱うデータはたとえば生徒がもらう成績表では国語が何点、数学が何点といったものがありますが、これは一次元のデータと言われます。

科目	点数
国語	
数学	
地理	
物理	

一次元データ、二次元データなど耳なれない言葉と思いますが、ひんぱんに使われるコンピュータ用語ですからできれば覚えてください。この種のデータを扱う時に使って便利なのが配列変数なのです。

配列変数？

この変数の特徴はプログラムで使用する前にDIM(ディメンジョン)文で大きさを宣言しておく必要があることです。

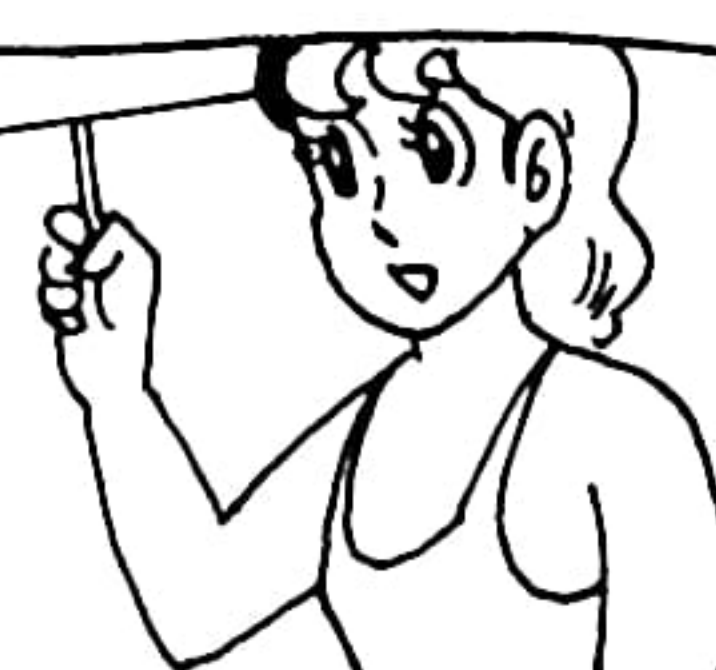
メモリ

DIM A(4)

たとえば DIM␣A(4)と宣言すると、そのあとのプログラムで同じ名前で番号の違う5つの変数が使えるようになります。なぜ指定した4よりもひとつ多いのかと言うと0(ゼロ)番目もあるからです。

DIM␣A(4)

一次元

このあと
A(0)
A(1)
A(2)
A(3)
A(4)
の変数名が使える。

また DIM␣A(3、2)にすると、12の変数が使えるようになります。

サイズ

DIM␣A(3、2)

二次元

A(0、0) A(0、1) A(0、2)
A(1、0) A(1、1) A(1、2)
A(2、0) A(2、1) A(2、2)
A(3、0) A(3、1) A(3、2)

DIM␣A(11)で得られる12の変数とは用途が違います。

!

0から3までの限定変数

0から2までの限定変数

A(I、J)

A(0、0)からA(3、2)まで12種の変数

このデータを指定したことになる。

IとかJとかの中身は宣言したサイズの中にある数値です。

1 2

A(I、J)

プログラム

A(1,2)

A(1、3)

A(1、4)

またDIM(3)は I=3：DIM(I)としても同じです。なんでこんなめんどうなことをと思われるかも知れませんがFOR〜NEXT文と併用すればスゴイことができるのです。

その指定は、変数名の次の(　)の中で行ないます。

A(　)

ここで何番目のメモリかを指定します。

それぞれのメモリは、変数Aの何番目という形で指定できるようになります。

A

0 1 2 3

0番もある！

くりかえしますと、DIM A(3)を実行したあとは、Aという変数名で4つのメモリが与えられ、

A

RAMメモリ

この場合Aのメモリは0番から始まって3番目なので、4以上の数字を入れるとエラーとなります。

DIM A(3)

エラーになります

A(4)= A(2)+1

A(3)とすると、4番目のメモリが指定されます。

A

0 1 2 3

A(3)

たとえば、A(1)にすると2番目のメモリが、

A

0 1 2 3

A(1)

I=5

A(I)= 3.14

3.14

RAM

5

RAM

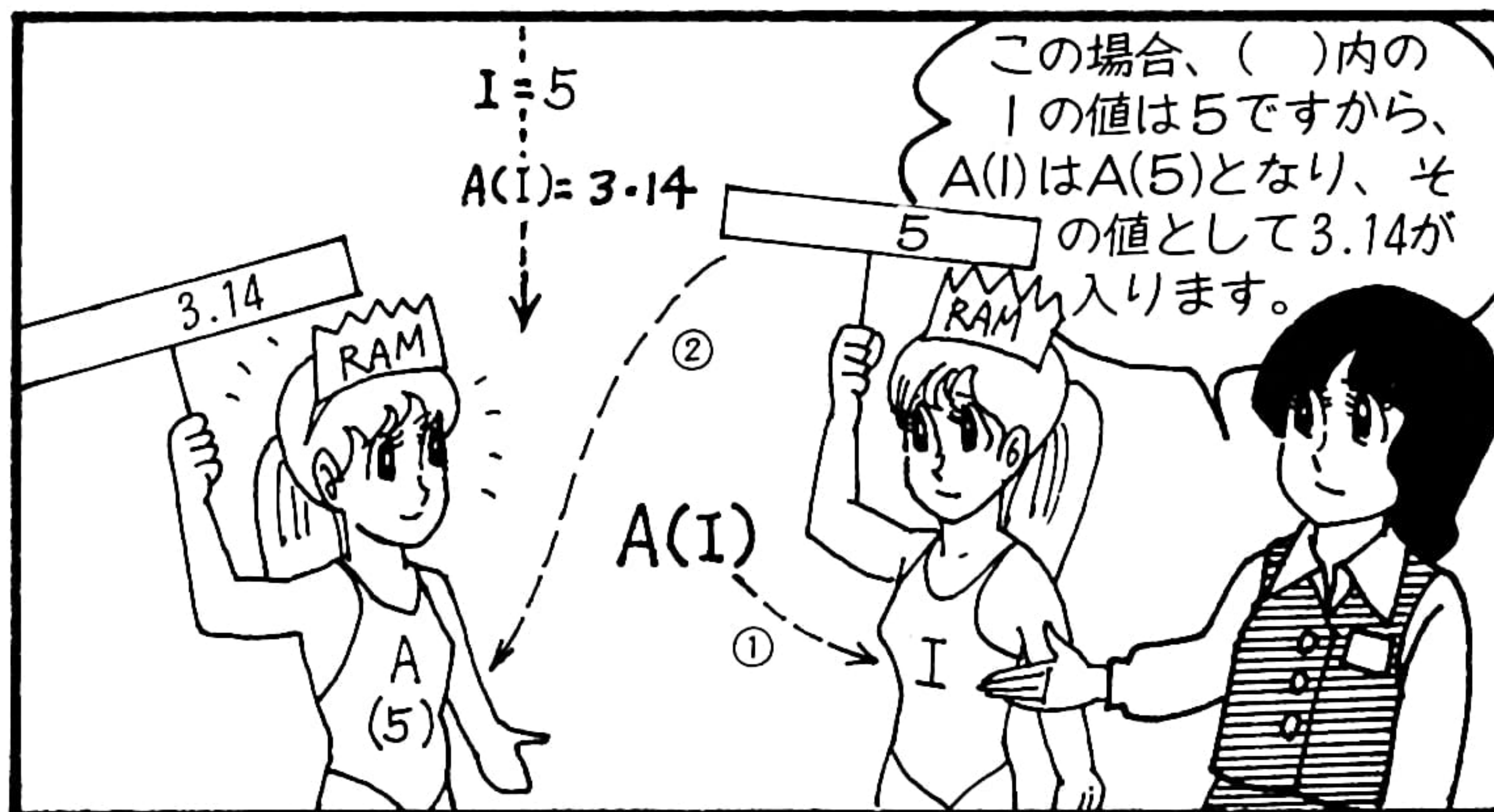
②

A(I)

①

A
(5)

I

この場合、(　)内のIの値は5ですから、A(I)はA(5)となり、その値として3.14が入ります。

この(　)の中は変数名を使っても、また式でもいいんです。

A(変数名)

A(式)

パソコンのメインメモリは、システムを動かすOSプログラム、BASICインタプリタ、私たちがつくるBASICソースプログラム、そして変数域から成り立っています。配列も変数域に割り当てられます。

メインメモリ

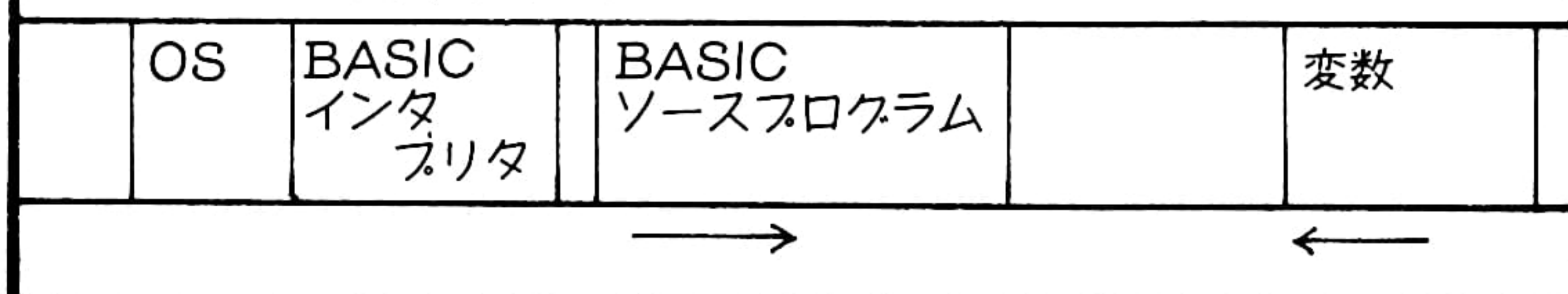
ROMまたはRAM

RAM

OS

BASIC インタプリタ

BASIC ソースプログラム

変数

( )内が式の場合は、まず式を計算して答えを出し、その答えで何番かのメモリを指定することになります。

A(I+A(1))

まずここが計算される

配列はデータをひとまとめにしてメモリすることによってFOR~NEXT文でうまく処理できるようになっています。

```
FOR I=1TO N
X=X+A(I)
NEXT I
```

たとえば、インプットの時は次のようにできます。

プログラム例です。

```
10 DIM A(3)
30 WAIT 50
50 FOR I=0 TO3
60 PRINT "I=";USING"##";I
70 INPUT " A(I)=" ; A(I)
80 NEXT I
```

プログラムを説明します。DIM A(3)は配列の宣言文です。メモリに割り当てた内容を書いておきましょう。

正義君のテストの成績

A

0	1	2	3
国語	数学	地理	物理
83	95	88	98

スゴイ！

WAIT50は表示する時間を指定するものです。ポケコンPC-1500の場合は、これが必要です。

時間を限定しないと、ここでプログラムが止まってしまう。

PRINT "I=" ; I

PRINT 'I=';USING'##';Iこれは次の何番目のデータをインプットするのか確認できるようにするためのものです。

'I='

I

I= 0

'##'

INPUT 'A(1)=';A(1)この文によってI=0からI=3までA(0)、A(1)、A(3)にインプットしたデータが入ります。

'A(I)='

A(I)= 65

キー操作 6 5 ENTER

PC-1500 RAMメモリ

A

0	1	2	3
83	95	88	98

これでひとまずデータをポケコンの中に入れることができます。

0	1	2	3
国語	数学	地理	物理
83	95	88	98

このデータを使って、正義君の平均点を求めてみましょう。

$$X=\frac{A(0)+A(1)+A(2)+A(3)}{4}$$

これでもいいんですが、配列を使う意味が半減します。

やはりFOR~NEXT文を使います。この方法もコンピュータで平均値を求める時のひとつの定石です。

```
100 X=0
110 FOR I=0 TO 3
120 X=X+A(I)
130 NEXT I
140 X=X/4
```

／（スラッシュ）割算の記号です。

プログラムを説明します。X=0、これはFOR～NEXTの中でXに累計値を入れていくので、最初にクリアしてXをゼロにしておく必要があります。

0
X=0
FOR
NEXT
X

次のX=X+A(I)これが累加するための文です。

(X) ① X+A(I) ② 計算 ③
A(0) 83
A(1) 95
A(2) 88
A(3) 98
RAM
X

プログラムをRUNすると、FOR～NEXTにはさまれたX=X+A(I)は、次のように変化します。

I=0で
X=X+A(0)
↑ ↓ ↓
83 0 83
0
RAM
X

I=1で
X=X+A(1)
↑ ↓ ↓
178 83 95
83 178 95
RAM
X A(I)

I=2で
X=X+A(2)
↑ ↓ ↓
266 178 88

I=3で
X=X+A(3)
↑ ↓ ↓
364 266 98
364
RAM
X

となり、FOR～NEXTを抜ける時にはXには364が入っています。

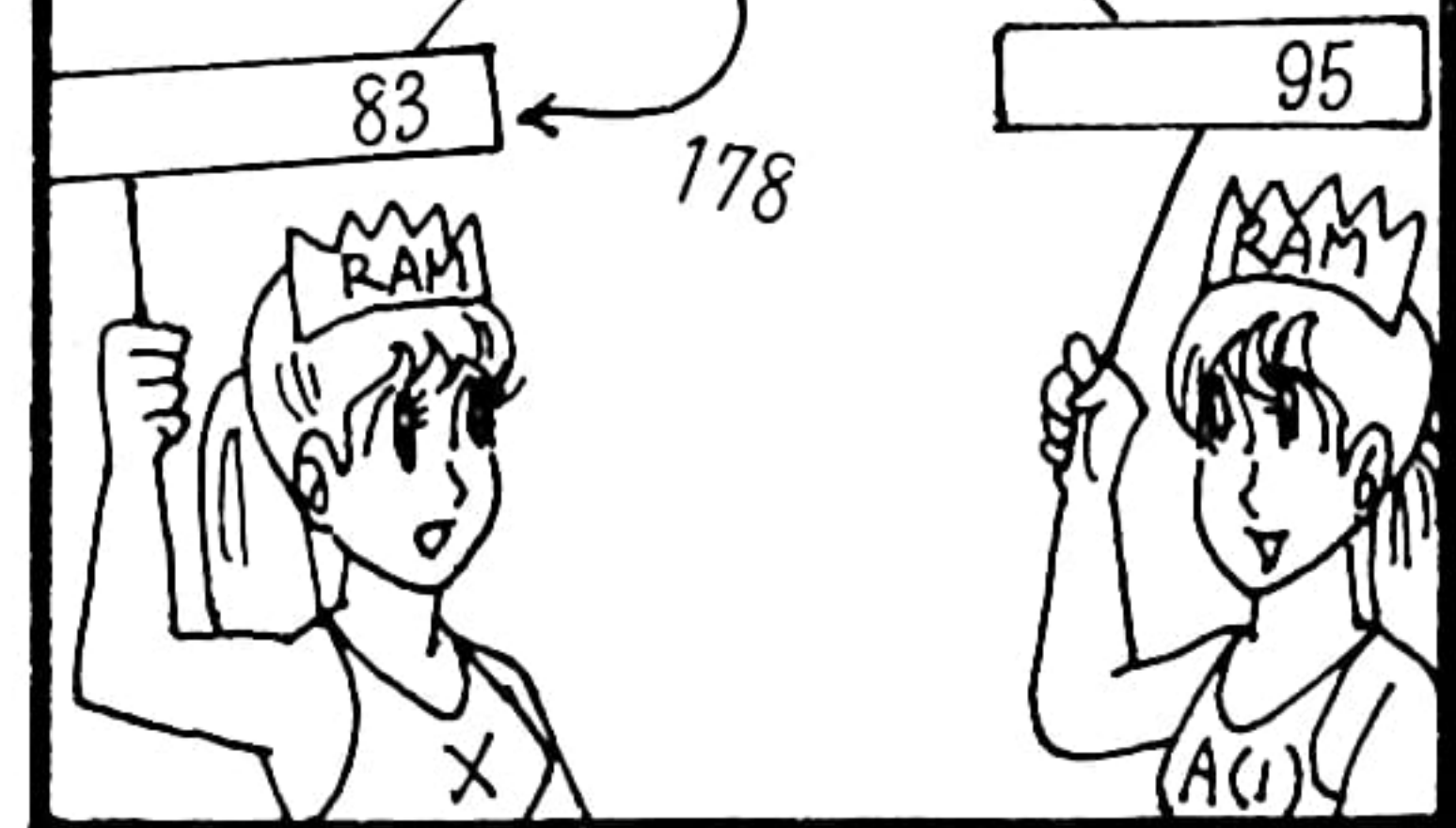
このXを4で割ると平均値が求められます。つまりX=X/4です。

364
X/4
91
計算
RAM
X

ただ、合計も印字したいような場合は別の変数名を使ってXの内容をくずさないようにします。

X/4が入る
91
変化しない
364
X/4
計算
Y=X/4
RAM
Y X

たとえば、Y=X/4とすると、Yに平均値が入り、Xは元のままで変化しません。

X：合計
Y：平均値

何度も出てきましたが、プログラムの式の左辺は、ひとつのメモリを示していて右辺の計算の結果をそのメモリに移せということなのです。移し終わると、左辺の内容は変わっています。

X=X+A(I)

ですから、左辺に計算式があるというのは間違いです。左辺は右辺の計算結果を入れるところです。

X ⇐ X+A(I)
↑ ↑
計算後 計算前

あとは、これをフォーマットを決めてプリントすればいいですね。

A(0)=␣␣␣␣␣
A(1)=␣␣␣␣␣
A(2)=␣␣␣␣␣
A(3)=␣␣␣␣␣
HEIKIN=␣␣␣␣␣

プログラム例です。

```
200 FOR I=0 TO 3
210 LPRINT "A(";USING
    "##";I;")=";
    USING "####";A(I)
220 NEXT I
230 LPRINT "HEIKIN=";X
```

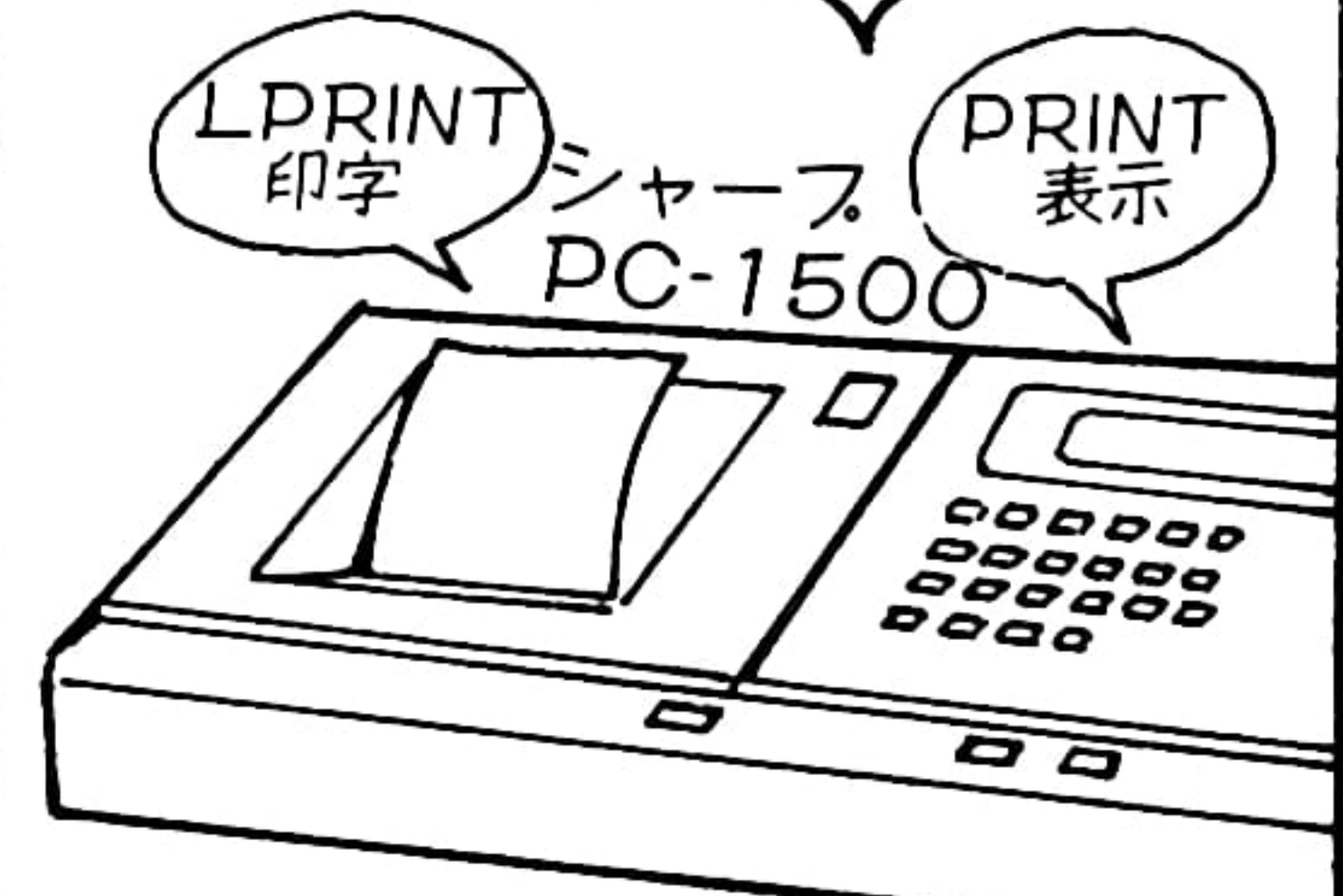
次のUSING"####";A(I)で、得点をプリントします。これがI＝0から3まで4回繰り返されます。

A(␣0)=␣␣83
A(␣1)=␣␣95
A(␣2)=␣␣88
A(␣3)=␣␣98
符号　符号

その次にフォーマットが続き"A("USING"##;I;")="でA(I)= とプリントしようとしているのです。

"A(";USING"##";I;")=";
A(␣␣)=
この1桁は正負の符号に使われる

プリントプログラムの説明をします。LPRINTはPC-1500のプリンタに出力せよという命令です。

LPRINT 印字
シャープ PC-1500
PRINT 表示

このプログラムで、正義、悦子、拓人の3人の平均点を求めました。

T　E　M

```
TOKUTEN
NAME=MASAYOSI
A( 0)=  83
A( 1)=  95
A( 2)=  88
A( 3)=  90
HEIKIN=  89

TOKUTEN
NAME=ETUKO
A( 0)=  85
A( 1)=  98
A( 2)=  89
A( 3)=  92
HEIKIN=  91

TOKUTEN
NAME=TAKUTO
A( 0)=  88
A( 1)=  99
A( 2)= 100
A( 3)=  96
HEIKIN=  95
```

```
STATUS 1
            253
 10:DIM A(3)
 20:INPUT "NAME=";
    A$
 30:WAIT 50
 50:FOR I=0TO 3
 60:PRINT "I=";
    USING "##";I:
    BEEP 1
 70:INPUT "A(I)=";
    A(I)
 80:NEXT I
100:X=0
110:FOR I=0TO 3
120:X=X+A(I)
130:NEXT I
140:X=X/4
180:LPRINT "TOKUTE
    N"
190:LPRINT "NAME="
    ;A$
200:FOR I=0TO 3
210:LPRINT "A(";
    USING "##";I;"
    )=";USING "###
    #";A(I)
220:NEXT I
230:LPRINT "HEIKIN
    =";X
240:END
```

それではこのプログラムをPC-1500でインプットしてみます。

これがプログラムリストです。253バイトあります。NAMEをA$として誰の得点かわかるようにしました。BEEPは音を出す命令です。

しかし、配列として最も多く使われるのは次にお話する2次元の配列でしょう。

A(I)…一次元配列
A(I, J)…二次元配列

ここまでは1次元の配列についてのお話でした。

DIMENSION A(3)
A(0)、A(1)、A(2)、A(3)

ひとまとまりのデータを扱う時は、配列とFOR～NEXTを使って処理するのが一番です。

X＝X1＋X2＋X3……
単独の変数ではFOR～NEXTは使えない

二次元配列というのは、ソフトの面から言うと、2つの指定ができる部分を持った変数です。

A(I, J)

メモリの数は、(I＋1)×(J＋1)個になります。これは、I＝0、J＝0もあるためです。

変数名 A

I → 0 1 2 3

J ↓ 0 1 2 3 4

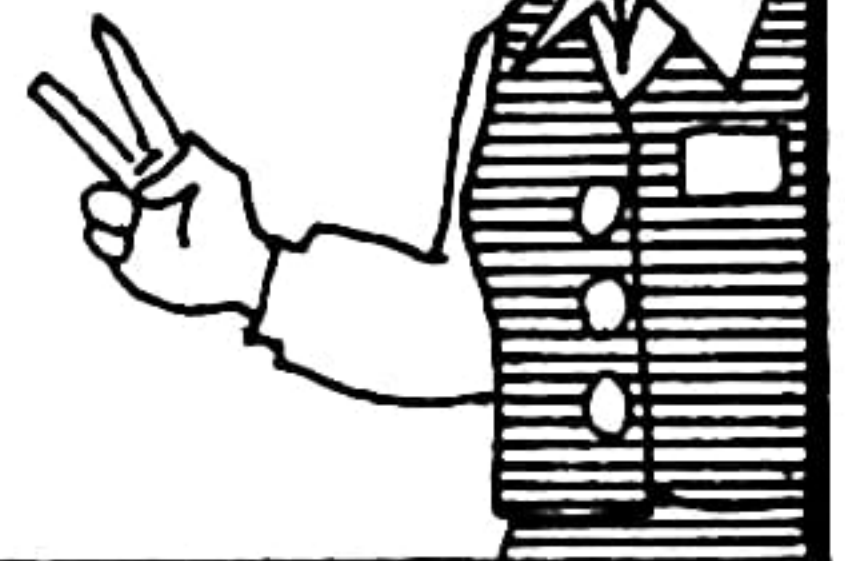
一見むずかしそうですが、よく見てください。普通の変数と同じなのです。

外見にまどわされないようにね。

A(I,J)

百聞は一見にしかず、とか。まず使い方の例を見てください。

この表は3人の得点をひとつにまとめたものです。

個人別平均
科目別平均
そして総平均
を求めます。

	正 義	悦 子	拓 人	平均A
国 語	83	85	88	
数 学	95	98	99	
地 理	88	89	100	
物 理	90	92	96	
平均B				

この表を横に計算すれば科目別平均が求められます。

答 答 答 答

また、縦に計算すれば個人別平均が求められます。

答 答 答

さらに、求められた平均を縦か横に計算することにより総平均が出てきます。

答

これに似た作表計算は、学校、会社に限らずいくらでも例があると思います。

パイパイ

会計

二次元配列では、Iを変化させると横の計算ができます。

A(0/1/2、J)

I → 0 1 2 3

J ↓ 0 1

また、Jを変化させると縦の計算ができるのです。

A(I、0/1/2/3)

I → 0 1 2 3

J ↓ 0 1 2 3

この番号10の文を実行した時、変数AXには、(3+1)×(4+1)=20のメモリが確保されます。

AX

(0,0) (0,1) (0,2) (0,3) …

まず、必要なメモリを確保するために配列の大きさを宣言します。変数名はAXとしましょう。

```
10 DIM AX(3,4)
```

変数名　配列のサイズ

そして変化させる手段がFOR〜NEXT構文というわけです。

```
FOR I=1 TO N
X=X+AX(I,J)
NEXT I
```

次に表のデータをこのAXの中にインプットする必要があります。

インプットの順として、個人別の科目の得点を入れることにすると表を縦に入れていくことになります。

そこでまずJを0から3まで変化させます。

```
FOR J=0 TO 3
PRINT"I=";USING"##";I;"J=";J:BEEP1
INPUT "AX(I,J)=";AX(I,J)
NEXT J
```

次にIを0から2まで変化させればよいのです。

```
FOR I=0 TO 2
  AX(I,J)
NEXT I
```

まとめて書くとこうなります。

```
30 WAIT 50
50 FOR I=0 TO 2
60 FOR J=0 TO 3
70 PRINT"I=";USING"##";I;"J=";J:BEEP1
80 INPUT "AX(I,J)=";AX(I,J)
90 NEXT J
100 NEXT I
```

これもよく使うパターンですよ。

インプットの順を間違えないようにしなくてはいけません。

次に平均値を求めるわけですが、まず累計しなくてはならないので、使うメモリを0にしておく必要があります。

AX(I,J)

J↓ \ I→	0	1	2	3
0				0
1				0
2				0
3				0
4	平均B 0	0	0	0

(列3の見出し：平均A)

AX(3、0)=0
AX(3、1)=0
AX(3、2)=0
⋮

クリアする時もFOR～NEXT文を使うとスマートになります。まず縦方向にクリアします。

J 0 1 2 3 4

1 2 3

```
FOR J=0 TO 4
AX(3,J)=0
NEXT J
```

次に横方向にクリアします。

0 1 2 3 4

I → 0 1 2 3

```
FOR I= 0 TO 2
AX(I,4)=0
NEXT I
```

文番号をつけてできあがりです。この番号は適当につけています。

```
150 FOR J= 0 TO 4
160 AX(3,J)= 0
170 NEXT J
180 FOR I=0 TO 2
190 AX(I,4)=0
200 NEXT I
```

この場合は10とびです。

まず横に累計するために I を0から2まで変化させます。この間Jは固定しています。

I → 0 1 2 3

J ↓ 累計

```
FOR I= 0 TO 2
AX(3,J)= AX(3,J)+AX(I,J)
NEXT I
```

このIが横累計のカギです。

1回横累計が終わるたびに今度はJを変化させます。これで科目別平均が順々に求められます。

I → 0 1 2 3

J 0 1 2 3 4

```
FOR J= 0 TO 3
AX(3,J)= AX(3,J)+ AX(I,J)
NEXT J
```

この部分が実行されるとメモリAX(3,J)には科目別累計が入っていることになります。

AX(3,0)
AX(3,1)
AX(3,2)
AX(3,3)

縦の場合も、説明は省きますが、同じ考え方でプログラミングできます。

I → 0 1 2 3

J ↓ 0 1 2 3 4

縦横まとめてプログラムしました。

```
300 FOR J= 0 TO 3
310 FOR I= 0 TO 2
320 AX(3,J)=AX(3,J)+AX(I,J)
330 NEXT I
340 NEXT J
350 FOR I =0 TO 2
360 FOR J =0 TO 3
370 AX(I,4)=AX(I,4)+AX(I,J)
380 NEXT J
390 NEXT I
```

横のカギ

縦のカギ

累計を縦・横それぞれの個数で割れば平均が求められます。

```
450 FOR J=0 TO 3
460 AX(3,J)=AX(3,J)/3
470 NEXT J
480 FOR I=0 TO 2
490 AX(I,4)=AX(I,4)/4
500 NEXT I
```

その平均値を縦に累計して4で割り総平均を求めればこの計算は終りです。

→ I

2 3 J↓

0 1 2 3 4 総平均

```
550 FOR J=0 TO 3
560 AX(3,4)=AX(3,4)+AX(3,J)
570 NEXT J
580 AX(3,4)=AX(3,4)/4
```

これで必要な答えはすべて配列AXの中に入っています。あとは、これをプリントすればよいのです。

LPRINT

ポケコンPC-1500のプリンタの桁数は18桁です。

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

プリントはプログラムの総決算です。

ただメモリの内容を打ち出しただけではだめですね。

誰が見ても良くわかるように工夫する必要があります。

得点表印字フォーマット

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T	O	K	U	T	E	N											
								M				E				T	
K	O	K	U	G	O	×	×	×	×	○	○	○	○	×	×	×	×
S	U	G	A	K	U	○	○	○	○	×	×	×	×	○	○	○	○
T	H	I	R	I	␣	×	×	×	×	○	○	○	○	×	×	×	×
B	U	T	U	R	I	○	○	○	○	×	×	×	×	○	○	○	○
K	O	B	E	T	U	␣	H	E	I	K	I	N					
M	A	S	A	Y	O	S	I	=	○	○	○	○					
E	T	U	K	O	␣	␣	␣	=	×	×	×	×					
T	A	K	U	T	O	␣	␣	=	○	○	○	○					
K	A	M	O	K	U	B	E	T	U	␣	H	E	I	K	I	N	
K	O	K	U	G	O	=	○	○	○	○							
S	U	G	A	K	U	=	×	×	×	×							
T	H	I	R	I	␣	=	○	○	○	○							
B	U	T	U	R	I	=	×	×	×	×							
S	O	U	H	E	I	K	I	N	=	○	○	○	○				

表中の○○○や×××はメモリ内容をプリントするところです。

␣はスペースです。桁合せのために入れることがあります。

```
 10:DIM AX(3,4)
 30:WAIT 50
 50:FOR I=0TO 2
 60:FOR J=0TO 3
 70:PRINT "I=";
    USING "##";I;"
    J=";J:BEEP 1
 80:INPUT "AX(I,J)
    =";AX(I,J)
 90:NEXT J
100:NEXT I
150:FOR J=0TO 4
160:AX(3,J)=0
170:NEXT J
180:FOR I=0TO 2
190:AX(I,4)=0
200:NEXT I
300:FOR J=0TO 3
310:FOR I=0TO 2
320:AX(3,J)=AX(3,J
    )+AX(I,J)
330:NEXT I
340:NEXT J
350:FOR I=0TO 2
360:FOR J=0TO 3
370:AX(I,4)=AX(I,4
    )+AX(I,J)
380:NEXT J
390:NEXT I
450:FOR J=0TO 3
460:AX(3,J)=AX(3,J
    )/3
470:NEXT J
480:FOR I=0TO 2
490:AX(I,4)=AX(I,4
    )/4
500:NEXT I
550:FOR J=0TO 3
560:AX(3,4)=AX(3,4
    )+AX(3,J)
570:NEXT J
580:AX(3,4)=AX(3,4
    )/4
600:LPRINT "TOKUTE
    N"
610:LPRINT TAB 8;"
    M    E    T"
620:LPRINT "KOKUGO
    ";USING "####"
    ;AX(0,0);AX(1,
    0);AX(2,0)
630:LPRINT "SUGAKU
    ";AX(0,1);AX(1
    ,1);AX(2,1)
640:LPRINT "THIRI
    ";AX(0,2);AX(1
    ,2);AX(2,2)
650:LPRINT "BUTURI
    ";AX(0,3);AX(1
    ,3);AX(2,3)
660:LF 1
670:LPRINT "KOBETU
     HEIKIN"
680:LPRINT "MASAYO
    SI=";AX(0,4)
690:LPRINT "ETUKO
      =";AX(1,4)
700:LPRINT "TATUTO
      =";AX(2,4)
710:LF 1
720:LPRINT "KAMOKU
    BETU HEIKIN"
730:LPRINT "KOKUGO
    =";AX(3,0)
740:LPRINT "SUGAKU
    =";AX(3,1)
750:LPRINT "THIRI
    =";AX(3,2)
760:LPRINT "BUTURI
    =";AX(3,3)
770:LF 1
780:LPRINT "SOUEIK
    IN=";AX(3,4)
790:END
```

インプット（50～100）
ゼロリセット（150～200）
横累計（300～340）
縦累計（350～390）
横平均（450～470）
縦平均（480～500）
横平均累計（550～570）
総平均（580）
プリント（600～780）

プリントプログラムの例です。

```
600 LPRINT "TOKUTEN"
610 LPRINT TAB 8;"M␣␣␣E␣␣␣T"
620 LPRINT "KOKUGO";USING"####";
                 AX(0,0);AX(1,0);AX(2,0)
630 LPRINT "SUGAKU";AX(0,1);AX(1,1);AX(2,1)
640 LPRINT "THIRI␣";AX(0,2);AX(1,2);AX(2,2)
650 LPRINT "BUTURI";AX(0,3);AX(1,3);AX(2,3)
660 LF1
670 LPRINT "KOBETU␣HEIKIN"
680 LPRINT "MASAYOSI=";AX(0,4)
690 LPRINT "ETUKO␣␣␣=";AX(1,4)
700 LPRINT "TAKUTO␣␣=";AX(2,4)
710 LF1
720 LPRINT "KAMOKUBETU␣HEIKIN"
730 LPRINT "KOKUGO=";AX(3,0)
740 LPRINT "SUGAKU=";AX(3,1)
750 LPRINT "THIRI␣=";AX(3,2)
760 LPRINT "BUTURI=";AX(3,3)
770 LF1
780 LPRINT "SOUHEIKIN=";AX(3,4)
```

プリント幅が足りないのでこんなフォーマットになりました。

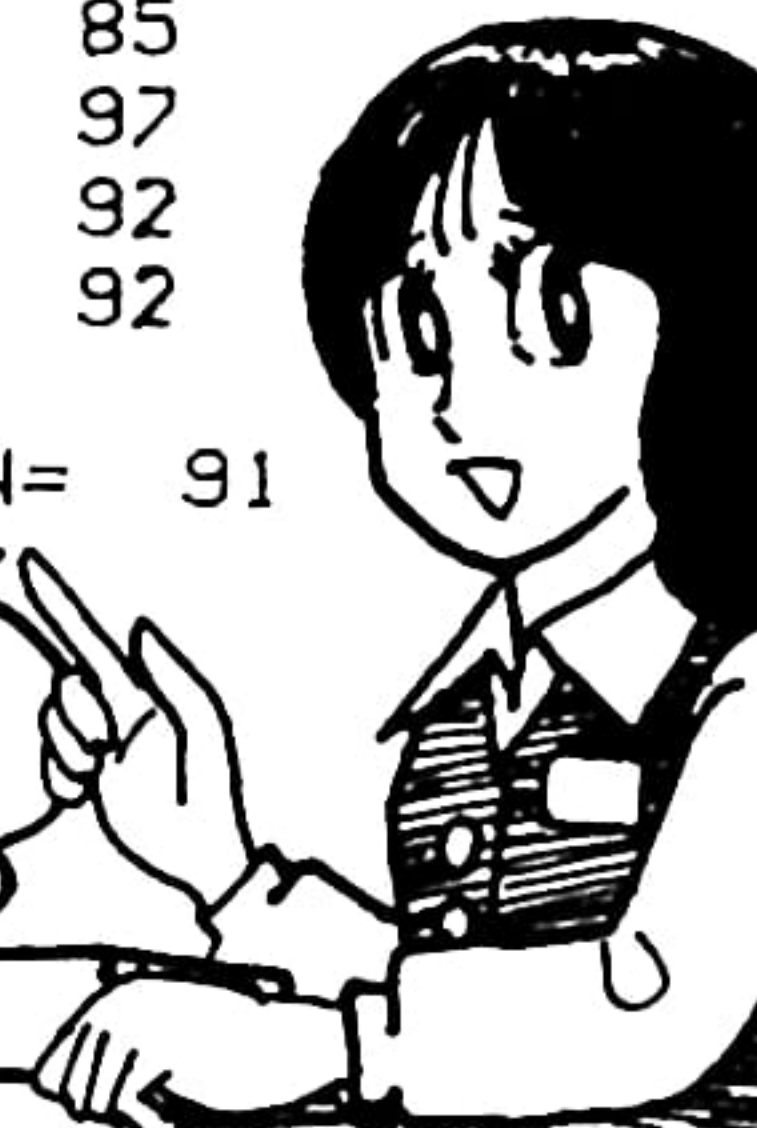
プリント結果です。

```
TOKUTEN
          M    E    T
KOKUGO   83   85   88
SUGAKU   95   98   99
THIRI    88   89  100
BUTURI   90   92   96

KOBETU HEIKIN
MASAYOSI=  89
ETUKO   =  91
TATUTO  =  95

KAMOKUBETU HEIKIN
KOKUGO=   85
SUGAKU=   97
THIRI =   92
BUTURI=   92

SOUEIKIN=   91
```

リストは次のコマを見てください。

LPRINT…
LPRINT…
LPRINT…
⋮

LPRINT…
LPRINT…
⋮

何回も繰り返してインプットする命令には、簡単にリピートできるように考えてあります。

PC-1500 ポケコンではリザーブ機能を使います。

一次元A(I)、二次元A(I,J)ときましたので、三次元のモデルを考えてみましょう。

A(I,J,K)

二次元の成績表にテストの回数の要素(K)を加えると、このように書くことができます。

生徒名 科目名 テスト回数

A(I、J、K)

	正義 I=0		悦子 I=1		拓人 I=2	
	前 K=0	後 K=1	前 K=0	後 K=1	前 K=0	後 K=1
国語 J=0						
数学 J=1						
地理 J=2						
物理 J=3						

さらにクラス別(L)といった要素を入れると四次元のモデルができます。

クラス

A(I、J、K、L)

L=0 L=1 L=2

Aクラス Bクラス Cクラス

この調子でいくと何次元の配列でもOKということになりますが、問題もあります。

A(I、J、K…
…X、Y、Z)

ひとつはインタプリタの能力です。たとえば、ポケコンPC-1500では、現在のところ配列のサイズは二次元まで、変数名は2桁までです。

DIM A(0、3、2)

ERROR

もうひとつはメインメモリの容量です。プログラムがまずメモリされ、次にプログラム実行時に変数域が大きいアドレスの方から形成されていきます。

プログラム 変数

メモリ余裕

プログラム実行時

配列で大きなサイズを宣言してしまうと、それだけプログラムのメモリできる領域が減ってしまうのです。

プログラム域

変数域

ERROR

そこで配列のサイズ、プログラムの大きさを考え、調整する必要がある時も出てきます。

また、いたずらに配列のサイズを大きくすることは考えものです。

適当にひとまとめにして、メインメモリから外部メモリにデータを移します。
カセットテープを使うことができます。

K=0 K=1 K=2 K=3

A(I,J) A(I,J) A(I,J) A(I,J)

レコード

メインメモリ

プログラム A(I,J)

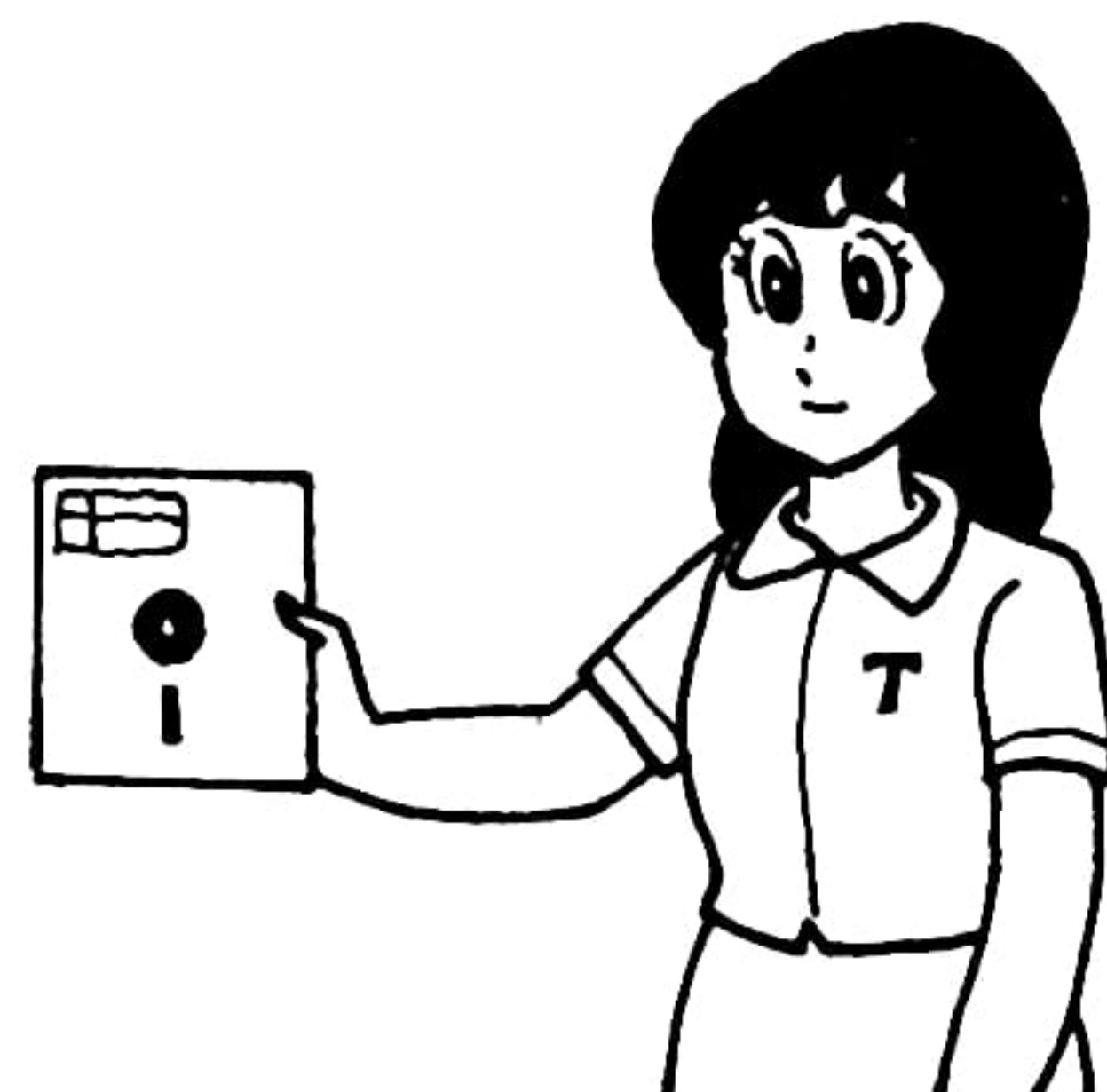
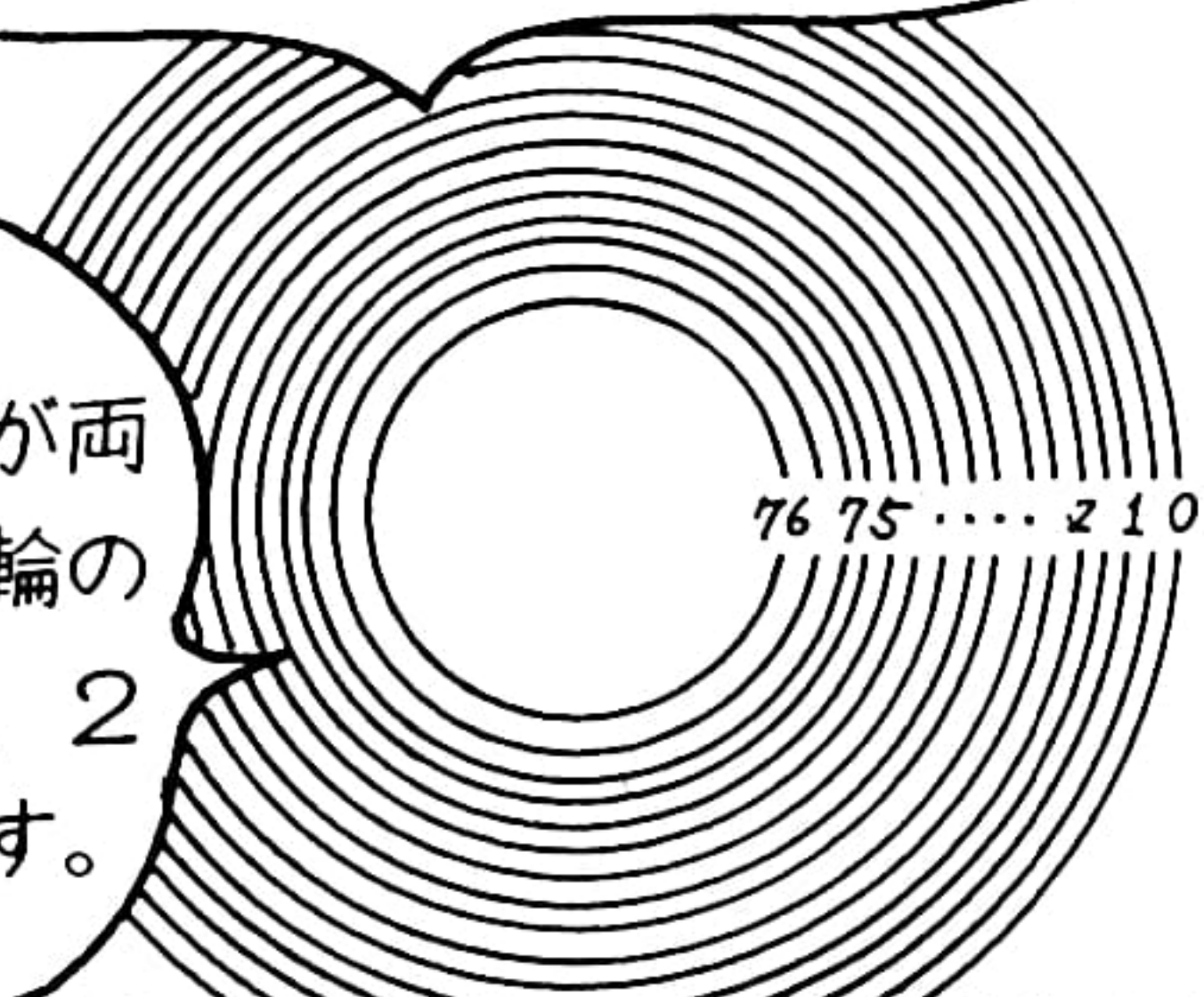
I=0 I=1 I=2

カセットテープ

⑥フロッピーに データファイルをつくる

フロッピーディスク（FDと略す）には3インチ、3.5インチ、5インチ、8インチのものがありますが8インチを例にとってお話しましょう。

8インチフロッピーの中には77個の輪が両面についています。輪の番号は外から0、1、2……76となっています。

76 75 …… 2 1 0

そこで8インチの場合は、これを計算すると1M(メガ)バイトの情報を記憶できるということになります。

256バイト/セクタ×
26セクタ/トラック×
2トラック/シリンダ×
77シリンダ=
1025024バイト=1Mバイト

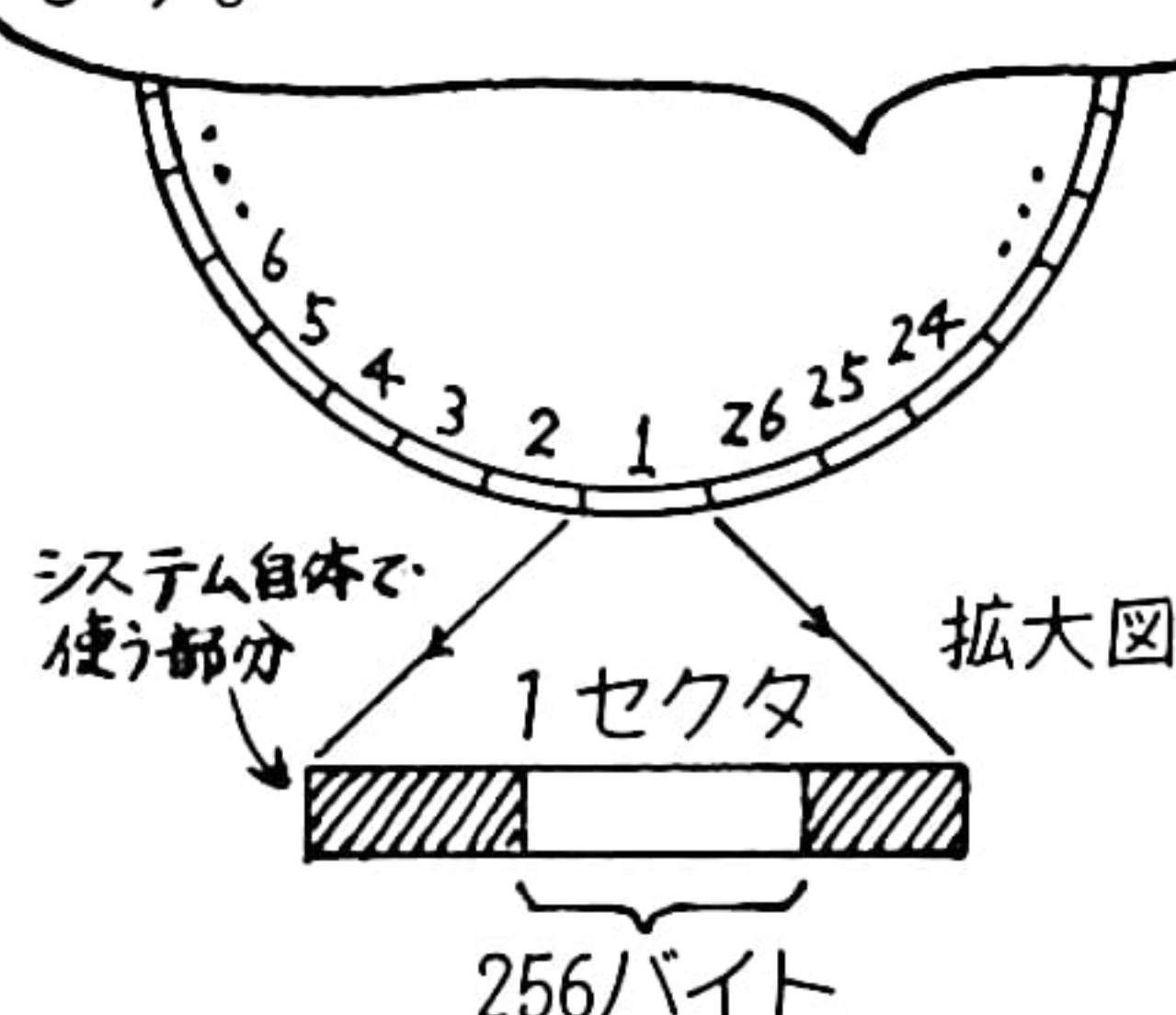
このトラックはさらに26コのセクタに区切られており、1セクタは256バイト記憶できます。

6 5 4 3 2 1 26 25 24

システム自体で使う部分

1セクタ

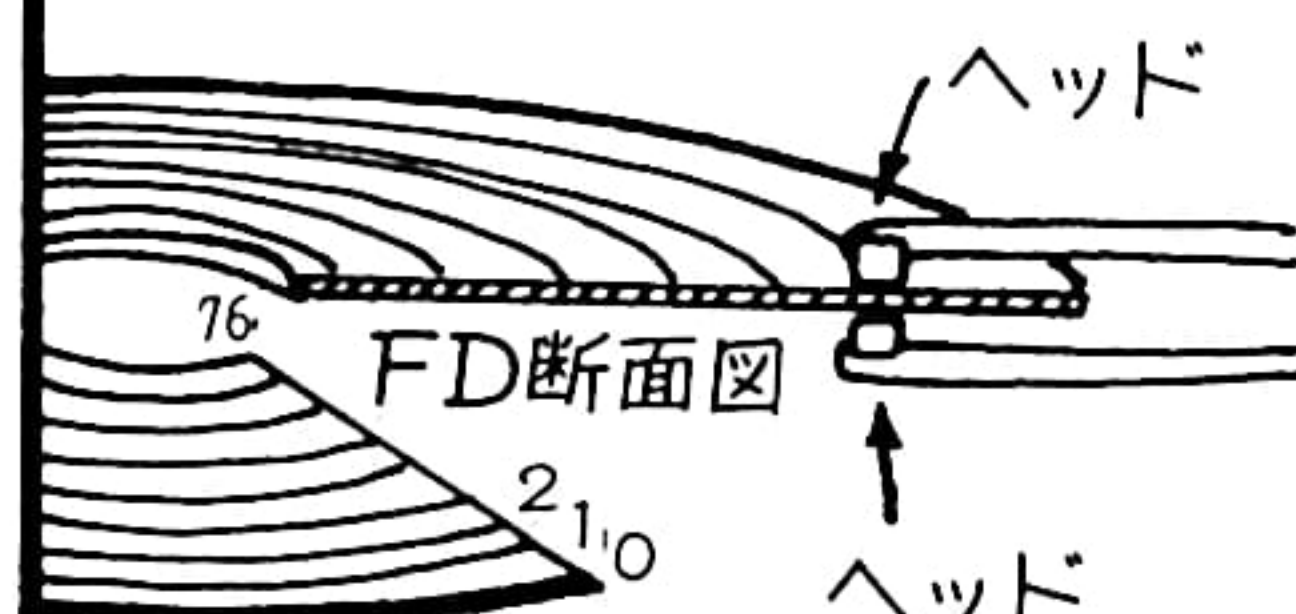
拡大図

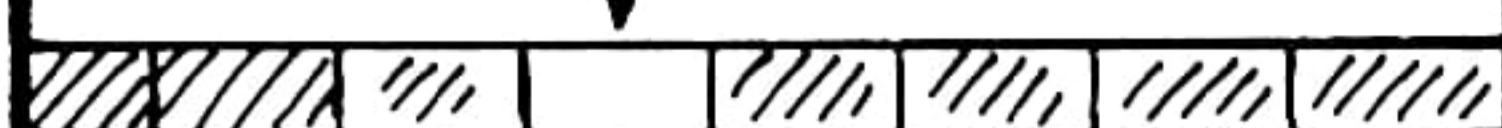
256バイト

この場合、77のシリンダがあると言い、1シリンダには2トラック（表と裏にひとつずつ）あります。ひとつの輪をトラックと言っています。

ヘッド

FD断面図

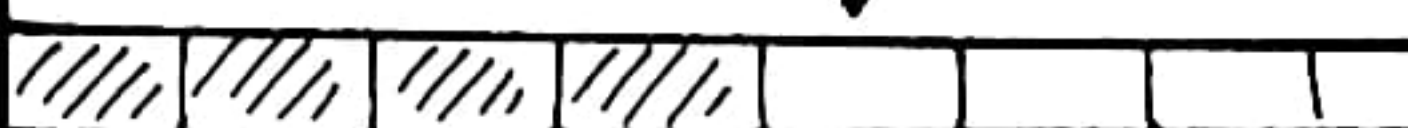
76 2 1 0

ヘッド

ランダムファイルは効率よく使えますがいろんな手続きがあってマニュアルを見ただけではむずかしいと思います。

あいている所に次のデータが書き込まれます。

ランダムファイル

フロッピーでつくるデータファイルにはシーケンシャルファイルとランダムファイルがあります。

順々に書き込まれます。

次のデータはここに書き込まれます。

シーケンシャルファイル

今まではプログラムのセーブやロードに使っていたフロッピーで、データのファイルをつくってみませんか。

データファイルはどう使うか？

プログラムファイルはSAVEとLOADで使える

おもしろそう。

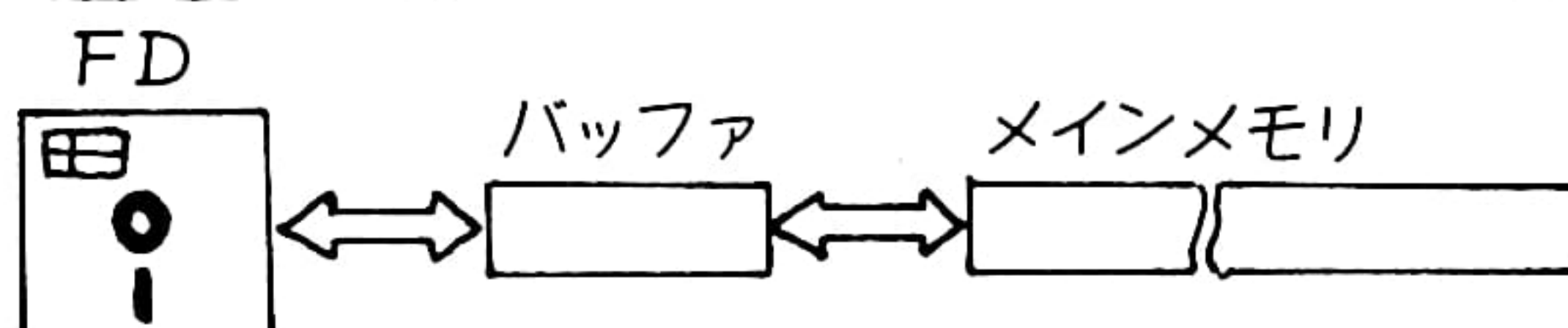
マニュアル

ここでマニュアルではもうひとつわからなかったことが明らかになるでしょう。さらに文字列の結合や分断もプログラムのテクニックにして使われています。程度は相当高いと思います。

また話の中に出てくるバッファというのは2つの速度の違うメモリの間においてデータの受渡しの調整をするところです。バッファレジスタとか言うことがあります。

FD

バッファ

メインメモリ

まず、OPEN文を使ってディスク・ファイルをオープンします。この文が実行されるとディスク使用状態になります。

OPEN "2:ETUKO" AS #1

そして、この文の意味は、

この説明のステートメントやファンクションはOKI-BASICのものです。他のパソコンでも考え方は同じです。

作成するファイル名をここではETUKOとすることになります。

1セクタ
1レコード
256バイト
ETUKO

ディスクドライブは＃2を使用し、

2

OPEN
"2:ETUKO"
AS #1

まず、バッファとして＃1（1番）を使用し、

＃1バッファメモリ
256バイト

ではFIELD（フィールド）文から説明します。これはファイル中のレコードのフォーマットを決めるものです。1レコードは256キロバイト、バッファと同じ長さです。

FIELD #n, a AS x1, b AS x2, ‥‥

ランダムファイル活用のためのプログラム文です。

RSET文
MKS$
PUT文
GET文
CVS
RIGHT$
LEFT$
MID$
FIELD

ばらばらでは、なんでこんな文があるのかわかりません。

次にCLOSE文がくるまではディスクドライブ＃2がOPENされたままになっているのです。

ディスクドライブ
I/Oポート

FIELD文は何回でも使うことができます。FIELD文はその変数と長さの割当表を作るようなものでデータの移動はありません。

FIELD #1, 20 AS A$, 30 AS B$, 10 AS C$
FIELD #1, 40 AS G$, 50 AS H$
FIELD #1, 128 AS J$, 128 AS K$

	1			2		3	
変数	A$	B$	C$	G$	H$	J$	K$
バイト	20	30	10	40	50	128	128

バッファに変数とその長さを割り当てることになります。

FIELD #1, 20 AS A$, 30 AS B$, 10 AS C$

A$に20バイト割り当てる

変数に割り当てるバイト数です。

#1
10バイト 30バイト 20バイト
C$ B$ A$

文字型変数です。

FIELD文によるバッファへの変数割り当てが完了したら、次はそれら変数の中身をバッファに移さねばなりません。

P$のデータ
TUKUTO
#1
20
FIELD
A$を使う
A$ B$ C$
20 30 10
P$

プログラムで扱うデータをバッファに移す命令がLSET文とRSET文の2つです。

RSET
LSET

LとRは左と右のこと、割当てたバイトの左右どちらかに詰めてデータをバッファに移します。

LSET A$＝P$
← 左づめ
TAKUTO
⇧14文字文のアキ
TAKUTO
P$

RSET B$＝Q$
B$ A$ #1
MASAYOSI TAKUTO
28字文のアキ
→ 右づめ
MASAYOSI
Q$

ここで注意が必要なのは、ランダムファイルはすべて文字型変数で取り扱われるということです。

P$ Q$

そこで数値をデータとしたい時にはまず数値を文字に変換しておく必要があります。

R$ R

この数値→文字変換がMKS$関数です。

R$＝MKS$(R)

これは make single $の略です。

MKS$関数では数値が4バイトの文字に変換されます。

R$＝MKS$(R)
;°G
3.123
R$
R

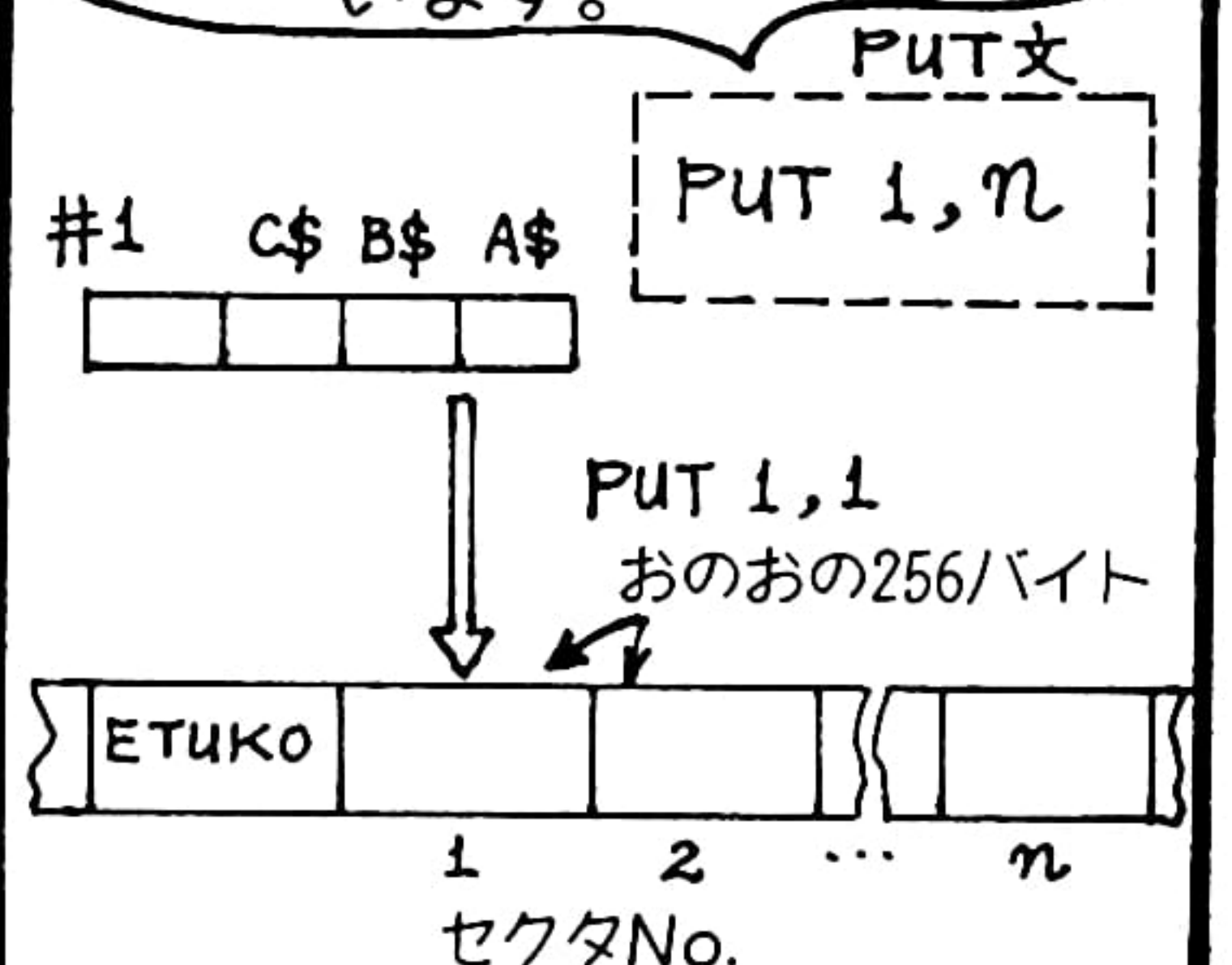
これで文字変換されたR$の内容をバッファにセットできます。このR$はC$にセットしたいと思いますが、C$の10バイトは本当は4バイトで十分なのです。

RSET C$＝R$
C$ B$ A$
;°G MASAYOSI TAKUTO
;°G
R$

これでバッファに移したデータをフロッピーに書き込む用意ができました。PUT文を使います。

PUT文
PUT 1, n
#1 C$ B$ A$
PUT 1,1
おのおの256バイト
ETUKO
1 2 … n
セクタNo.

PUT 1、1が実行されると、ETUKOというファイルの1番目のセクタにバッファの内容が移されます。

#1
C$ B$ A$
196バイトはアキ
10バイト 30バイト 20バイト
ETUKO ;°G MASAYOSI TAKUTO
ファイル名
1セクタ＝256バイト

今、その範囲を越え、プログラムの中で使用するデータをランダムファイル形式でフロッピーに書き込めるようになったわけです。

	数学	国語	物理	化学
正義				
悦子				
拓人				

今までフロッピーに書き込んでいたのは、いろんな形式のプログラムだけでした。

```
SAVE "A"
LOAD "A"
```

これでTAKUTO、MASAYOSHI、3.123という3つのデータをフロッピーに書き込んだわけです。

まずGET文でファイルからバッファにデータを移します。もちろん、その前に、ファイルはオープンされFIELD文でバイトの割当てが完了していなくてはなりません。

```
OPEN "2:ETUKO" AS #1
FIELD #1,20 AS A$,
 30 AS B$,10 AS C$
GET 1,1
```

では、次にこの3つのデータをプログラムで取り出してみます。

ただ今のやり方では大量の数値をデータとして扱うのには不便な気がします。たとえば、数値(4バイト)を256バイトにどのように割り当てていったらよいかということです。へたをすると、長い長いFIELD文になってしまいます。

バッファ #1

4 4 4 4 4 …

A$ B$ C$ D$ E$ ….

この問題は、またあとで検討したいと思います。

この時、それぞれのデータがどんな形式で入っているのかわかっていないとうまくいきません。

あとは、バッファ#1から必要なデータを取り出せばよいわけです。

C$…10バイト 右づめ　B$…30バイト 右づめ　A$…20バイト 左づめ

;°G▊　MASAYOSI　TAKUTO

LEFT$　P$

TAKUTO

実は3.123という数字

GET文はバッファの番号と、ファイルのセクタ(PUTしたセクタ)番号を指定するだけです。

1 2 3

ETUKO

GET 1,1

#1

Q$=B$ とすれば Q$の中身は 22コのスペース(表示すると空白)と MASAYOSI(8文字)になります。

C$の中の数値は4バイトの文字として入っていますのでRIGHT $の他に、文字→数字変換のCVS 関数も必要です。ひとつにまとめて書いてみました。

C$

;°G▊

R=CVS(RIGHT$(C$,4))

3.123

元の数字に戻す。 R

同様に、MASAYOSIはB$30バイトの中に右づめで入っているわけですから、RIGHT $文で8文字取り出せばいいですね。

B$

MASAYOSI

Q$=RIGHT$(B$,8)

MASAYOSI

Q$

まず、TAKUTOはA$20バイトの中に左づめに入っています。これを取り出すにはLEFT$文が便利です。A$の左から6文字取り出すのは、LEFT$(A$、6)と書き、これを別の変数に入れればよいのです。

A$

TAKUTO

P$=LEFT$(A$,6)

TAKUTO

P$

ところが、データが数値の場合は、FIELD文での変数割当てが問題です。

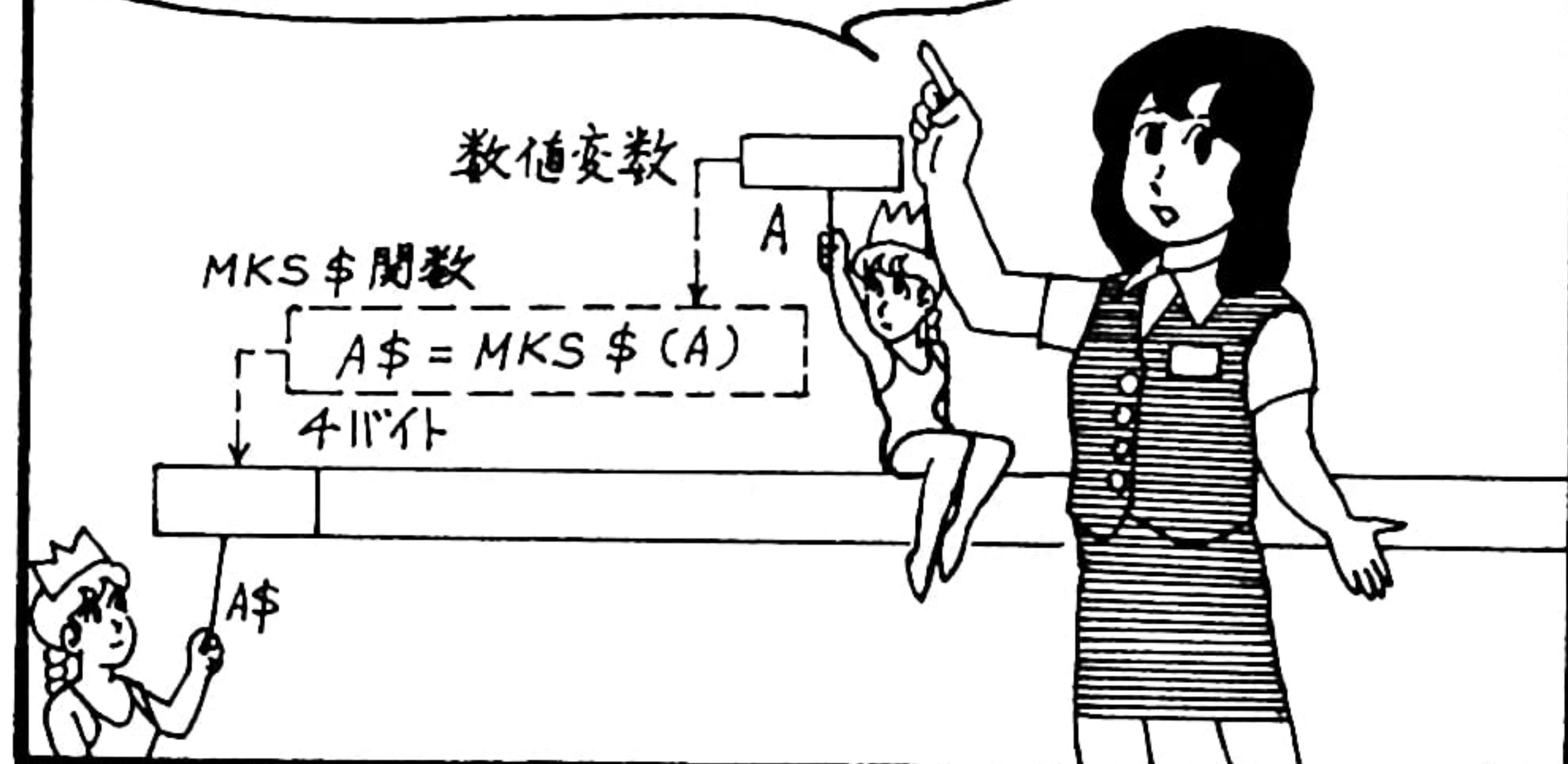
数値変数

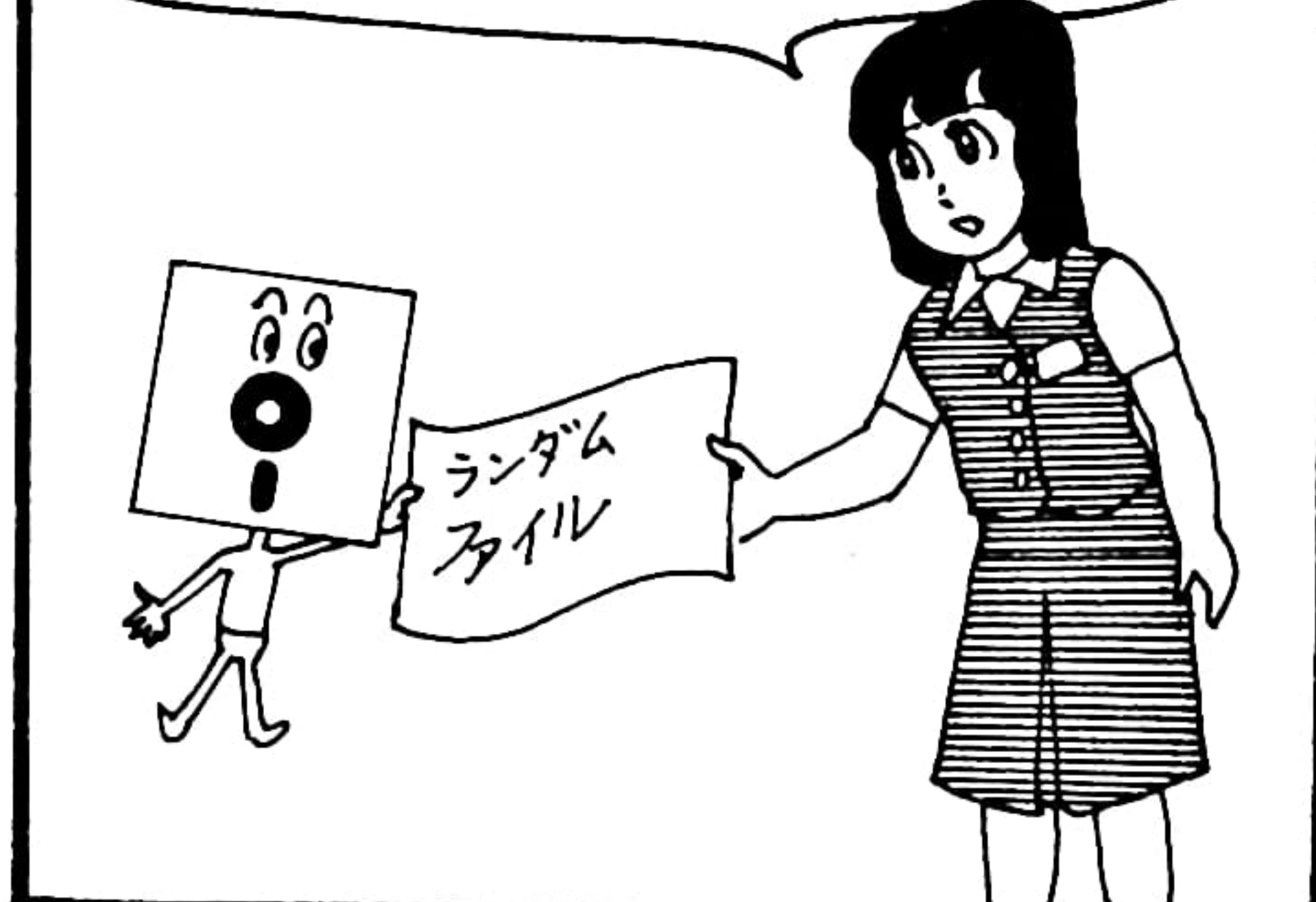
A

MKS$関数

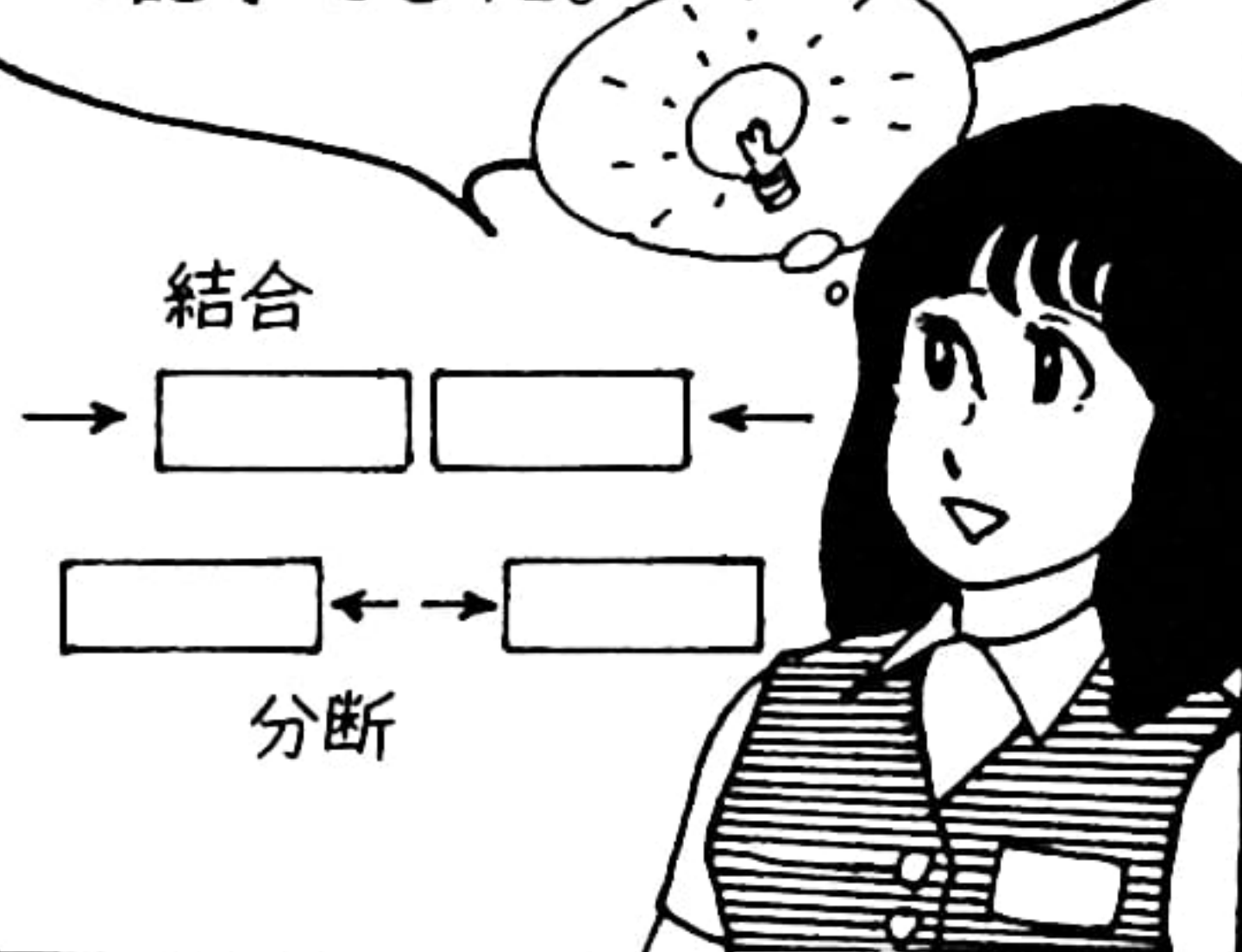
A$＝MKS$(A)

4バイト

A$

だいたいこんな感じで、フロッピーへのデータの読み書きができるわけです。

ランダムファイル

この時ふと思いついたのがポケコンのマニュアルに出ていた文字列の結合と分断の方法の記事でした。

結合

分断

しかし、64もの変数をどうやってFIELD文でバッファに振り当てたらいいのでしょうか。配列が使えなきゃ大変。

```
FIELD #1，4 AS A$
 4AS B$  4AS C$
 4 AS D$ …
```

64回もこれを書かなければならないなんて……。

このバッファに入れるのがすべて数値だとすると、256バイトを4バイトで割ればいいから、えーと

64個の数値が入れられます。

数字15.689も4バイトで表わせます。

4バイト

結合作業は配列変数を使わないと効果がありません。こんなぐあいにやります。

```
DIM A$(32)
N=32
FOR I=1 TO N
B$=B$+A$(I)
NEXT I
```

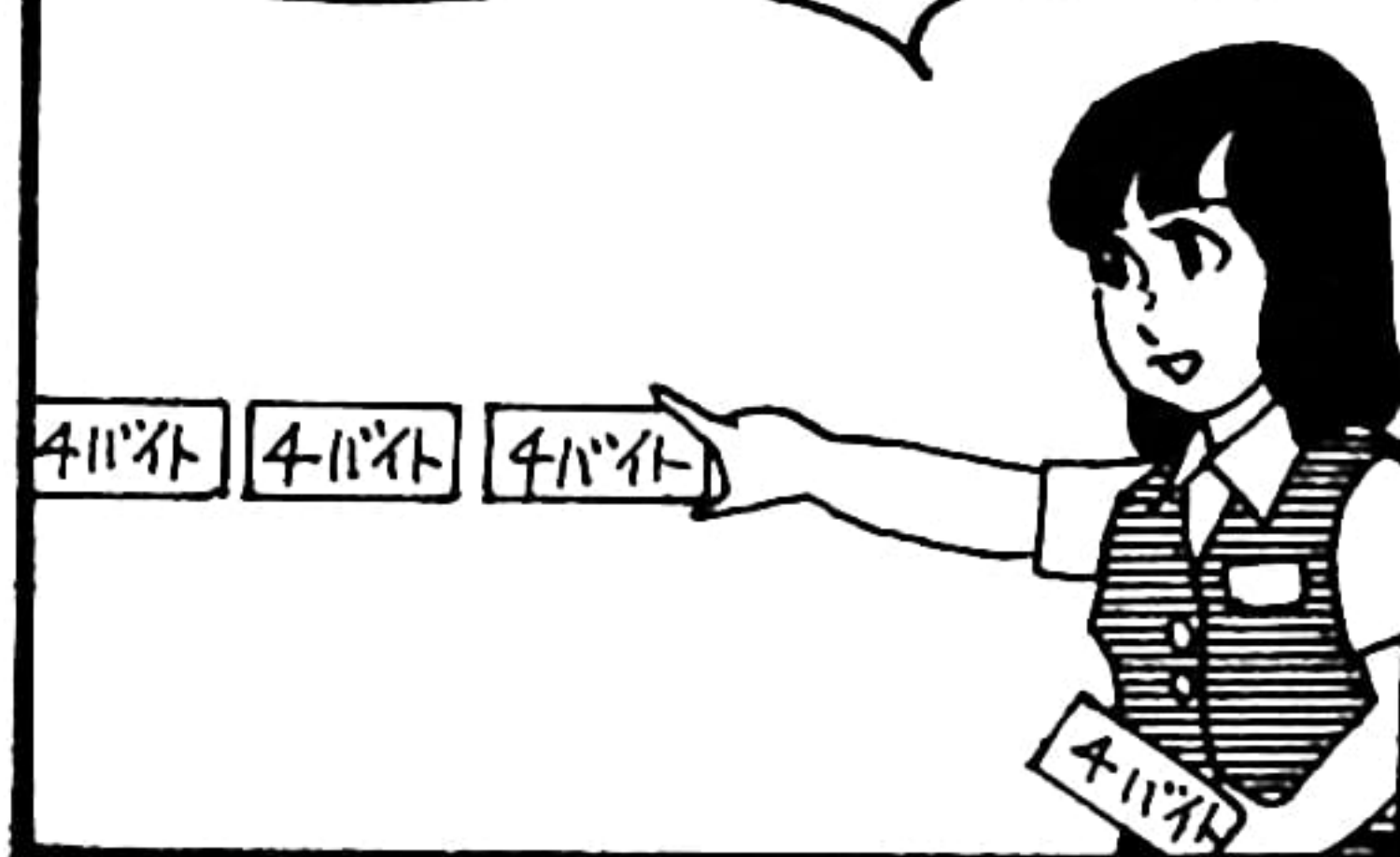
実行後B$はA$(I)がすべて4バイトとすると4×32バイトのなが～い文字変数になってます。

文字列の結合はすごく簡単です。

```
B$ = B$ + A$
```

このようなたし算でOKなのです。

それは、バッファを入れる時は4バイトの文字列をいくつかくっつけて大きな文字列とします。そうすればバッファに定義する変数の数が少なくてすみます。

4バイト 4バイト 4バイト

4バイト

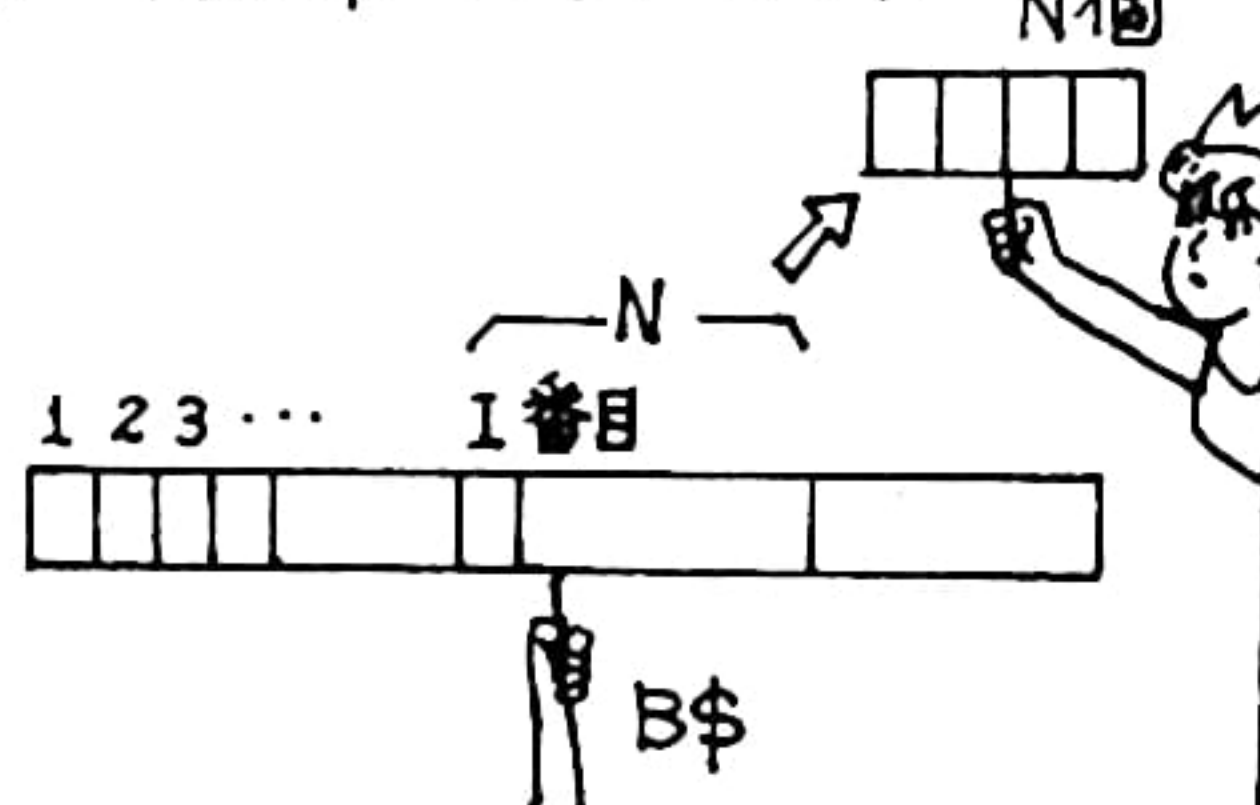
この時、包丁がわりに使わねばならないのがMID$関数です。文字列の左がわI番目からN個の文字を取り出すことができます。

```
A$ = MID$(B$,I,N)
```

N個

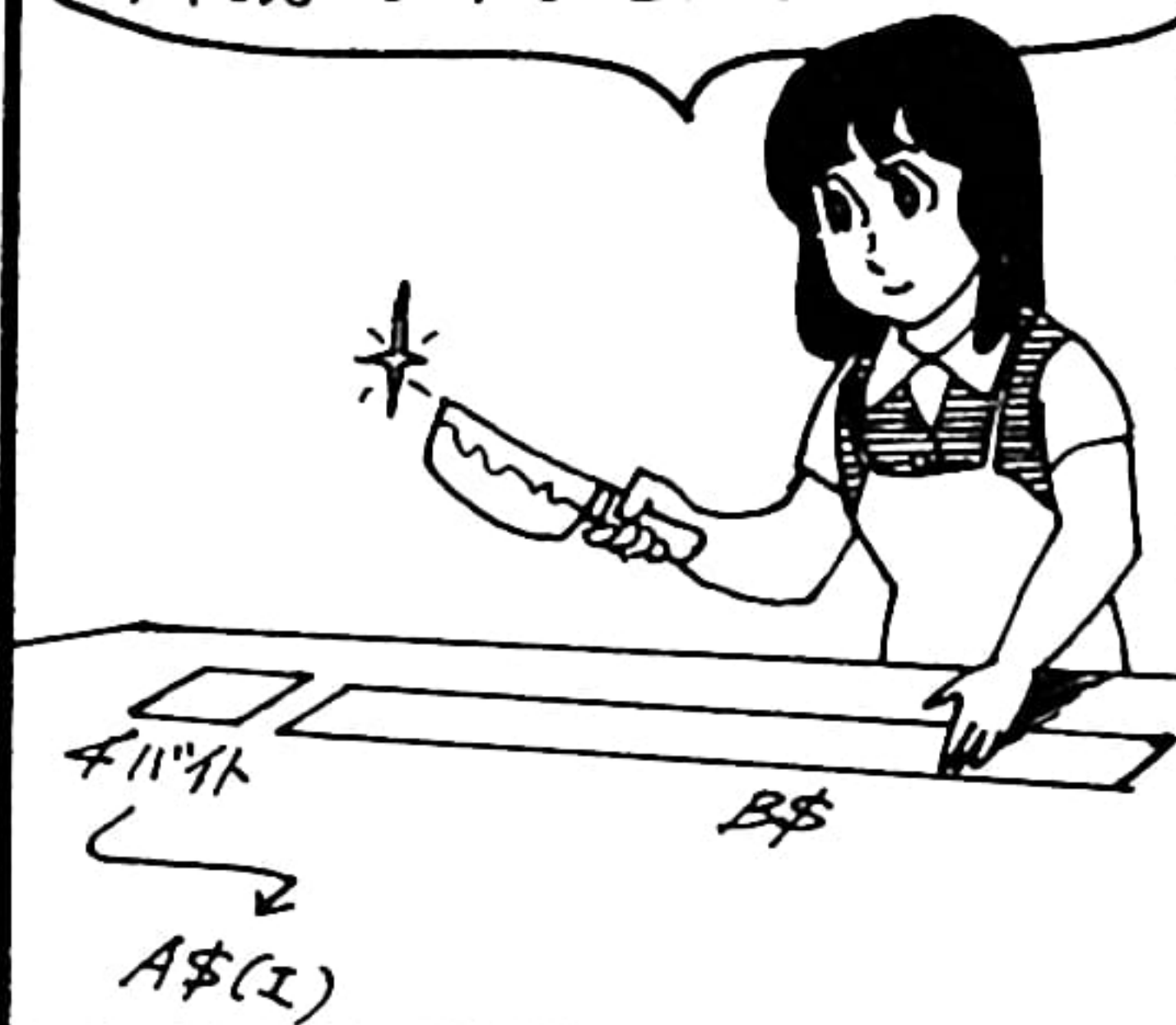
N

1 2 3 … I番目

B$

次に取り出す時は、このB$を4バイトずつに切り出してA$(I)に順番に戻してやらなければなりません。

4バイト

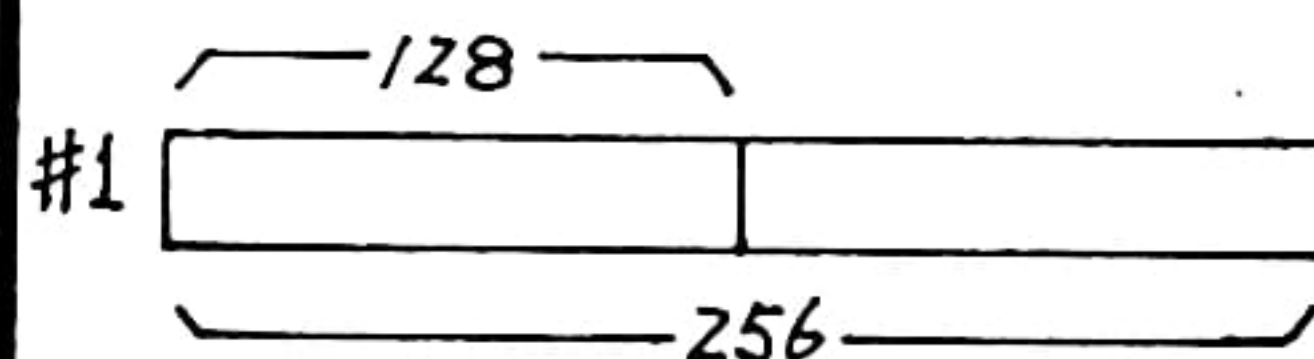
B$

A$(I)

このB$ひとつでバッファの半分を占めることになります。

```
FIELD #1,128 AS B1$
LSET B1$=B$
```

128

#1

256

B$は128バイト以下でなければなりません。

テストプログラムの内容
文字変数（左づめ、右づめ）と数値と配列値を一度フロッピーに書き込みます。
次に使った変数をクリアして消えたかどうか確認の表示をします。
次にフロッピーの内容を読み出してこれを表示してみようというものです。

では、読み書き確認のためのプログラムをつくって走らせてみましょう。

ランダムファイルで使うプログラム文や関数の役割がわかりました。

OPEN文
FIELD文
LSET文
RSET文
MKS$関数
PUT文
GET文
CVS関数
RIGHT$関数
LEFT$関数
MID$関数
CLOSE文

```
OPEN␣"1:ETUKO"␣AS␣#1
DIM␣S(32)
INPUT␣"NAME(L)=";P$
INPUT␣"NAME(R)=";Q$
INPUT␣"SUUJI=";R
INPUT␣"DATASUU␣N=";N
FOR␣I=1␣TO␣N
INPUT␣"DATA=";S(I)
NEXT␣I
```

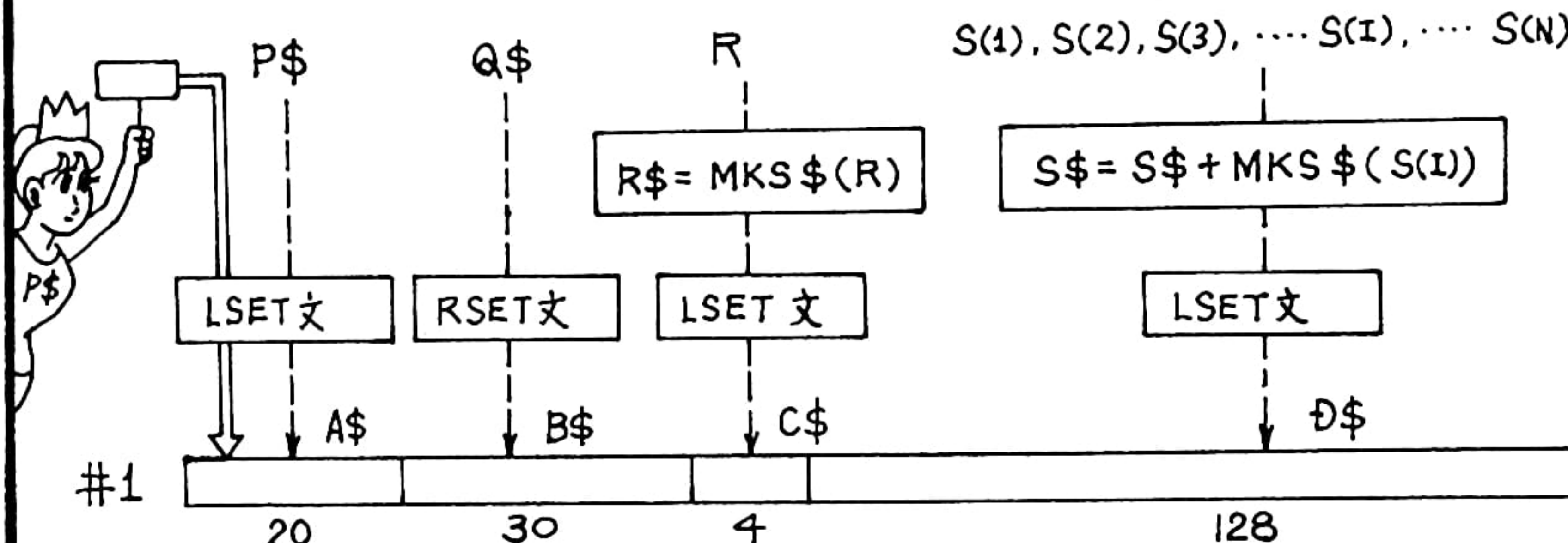
この部分をコーディングしてみます。ファイルのオープンから始めます。

␣はスペースコードです。キーインする時スペースキーをたたきます。

まずインプットのための変数をこのように割り当てました。

P$ …… 左づめ用
Q$ …… 右づめ用
R …… 数値
S(I) …… 配列
N …… データ数

FIELD␣#1,20␣AS␣A$,30␣AS␣B$,4␣A$␣C$,128␣AS␣D$

P$　Q$　R　S(1),S(2),S(3),……S(I),……S(N)
R$=MKS$(R)　S$=S$+MKS$(S(I))
LSET文　RSET文　LSET文　LSET文
A$　B$　C$　D$
#1　20　30　4　128

次はインプットした内容をバッファに移すまでの部分です。まずフィールド文でバッファの割り当てを定義します。

そして、ファイルへの書き込みは、PUT文です。

PUT␣1,2 セクタNo.
バッファ256バイト
#1
1:ディスクドライブ
256バイト
ETUKO　1　2 セクタ

```
R$=MKS$(R)
FOR␣I=1␣TO␣N
S$=S$+MKS$(S(I))
NEXT␣I
LSET␣A$=P$
RSET␣B$=Q$
LSET␣C$=R$
LSET␣D$=S$
```

数字→文字変換、文字列の結合、バッファへのデータ転送部分はこうなります。上の図と見くらべてソフト的な構造を確認してください。

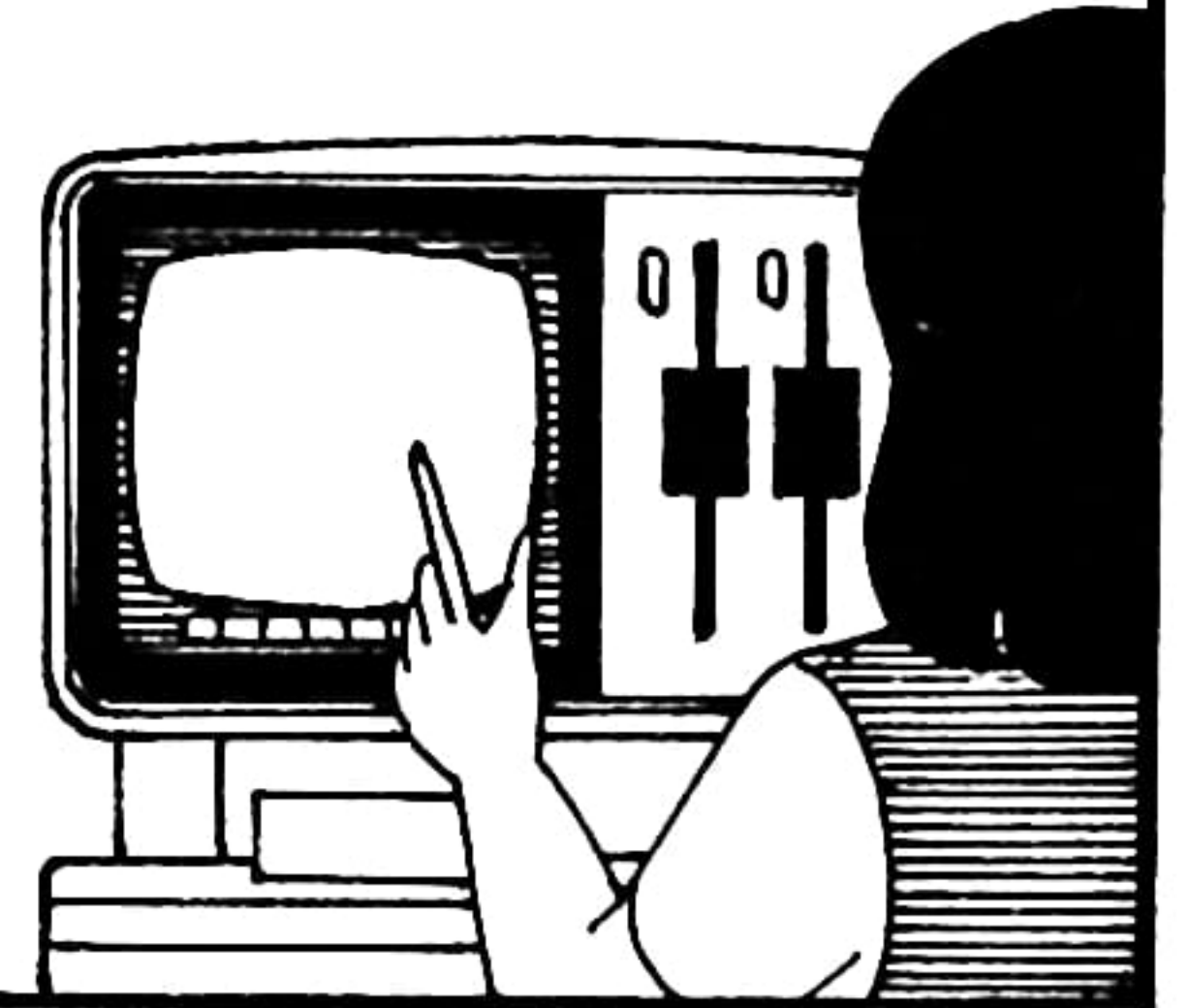
バッファへ書き込むにはすべて文字列でなければなりません。

```
P$=" "
Q$=" "
R=0:R$=MKS$(R)
FOR␣I=1␣TO␣N
S(I)=0
NEXT␣I
S$=" "
LSET␣A$=P$
RSET␣B$=Q$
LSET␣C$=R$
LSET␣D$=S$
```

また、バッファの内容と使った変数の内容をクリアしておきます。ここでCLOSEして電源を切ってやり直せばこんなことしなくてもいいんですけどね。

文字列のクリアは" "です。

ここでインプットの時の表示内容などをクリアしておきましょう。CLS文です。

```
CLS
```

ここでのハイライトはS$を4バイトずつに分解してS(I)に入れるところです。

```
P$=A$
Q$=B$
R$=C$
S$=D$
FOR␣I=1␣TO␣N
I1=1+4*(I-1)
S(I)=CVS(MID$(S$,I1,4))
NEXT␣I
```

I1=1+4*(I-1)

1 5 9 13

4 4 4 4

S$ S(I)

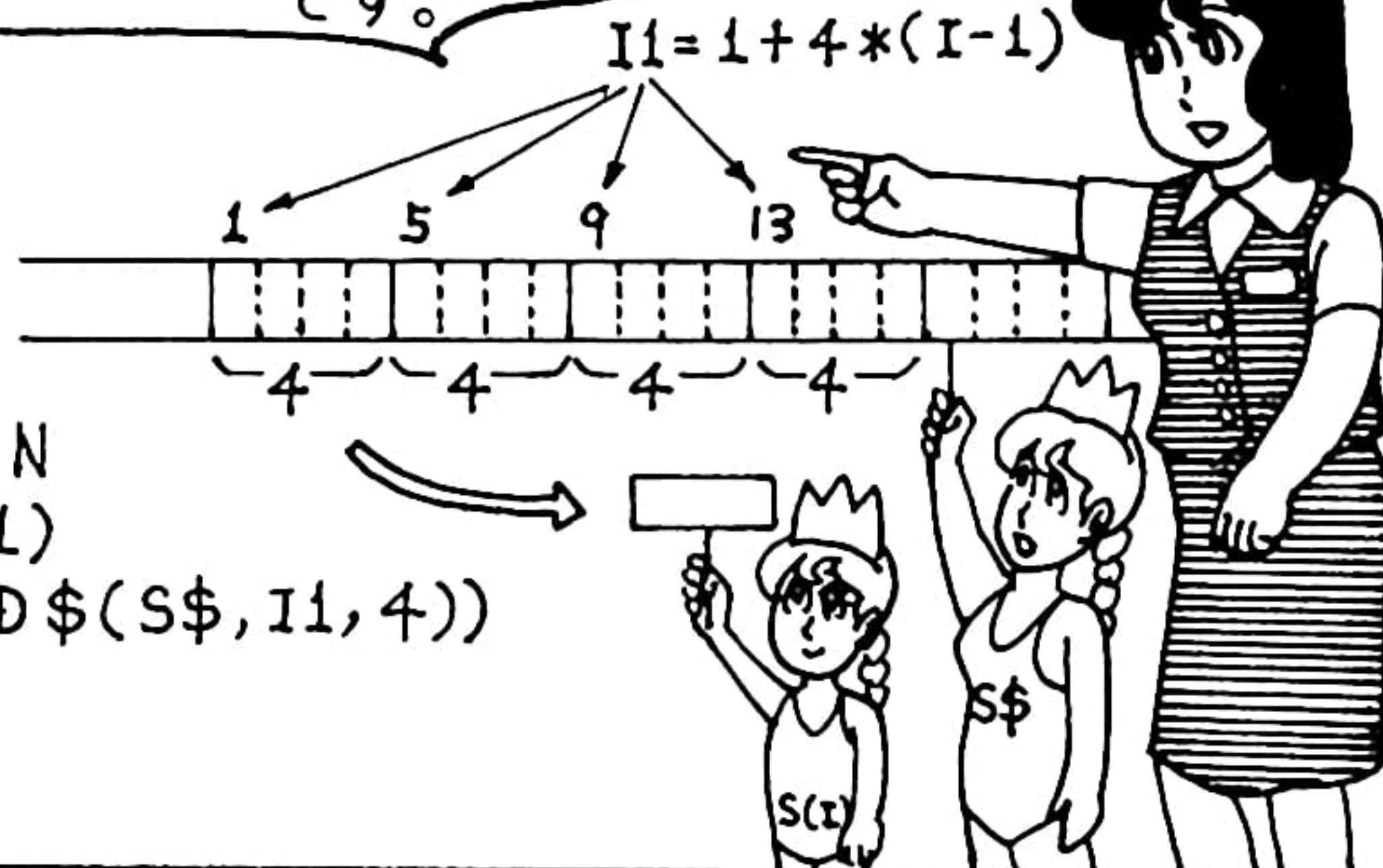
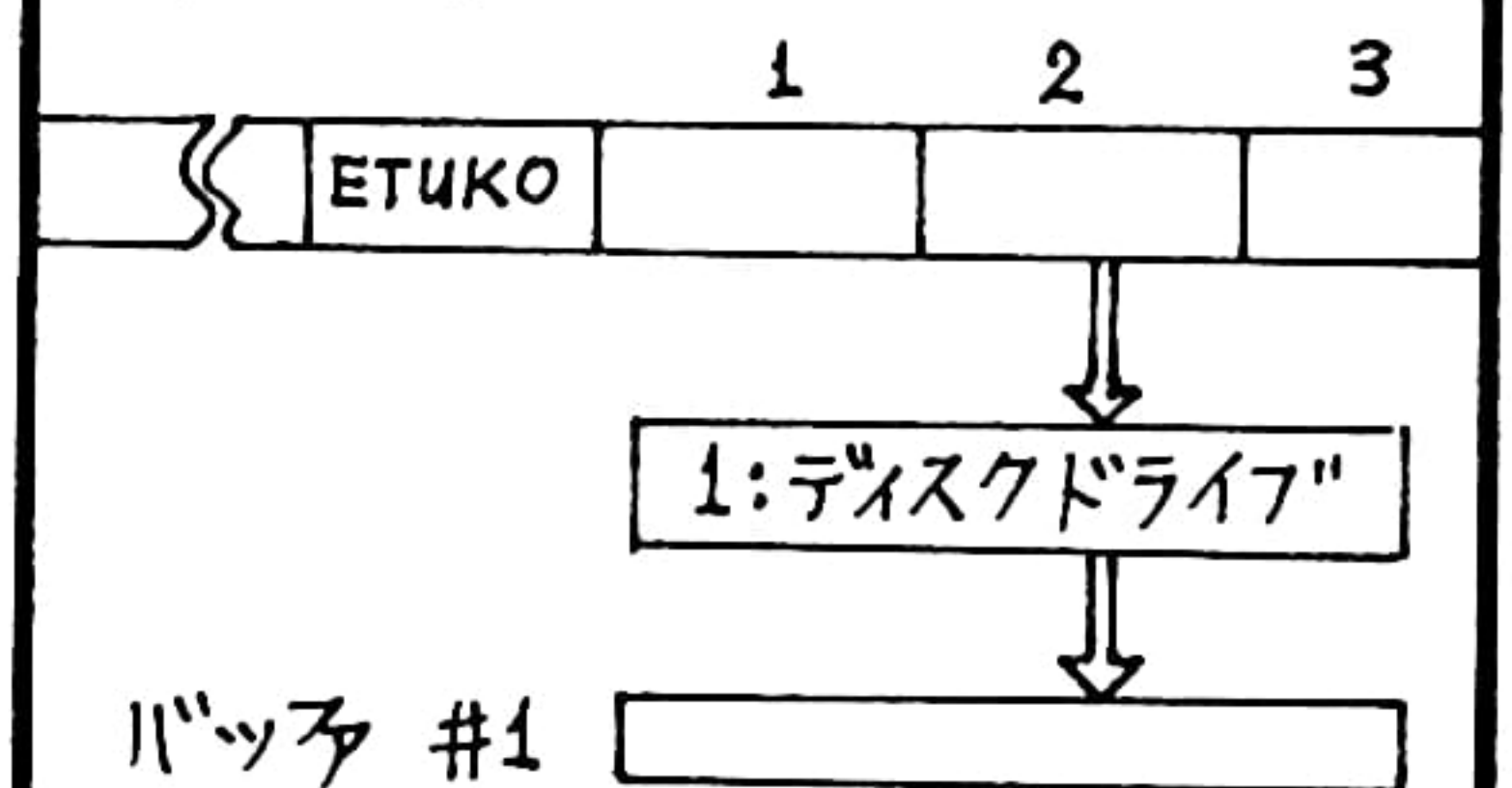
次にGET文でファイルからバッファへデータを読み込みます。

```
GET␣1,2
```

1 2 3

ETUKO

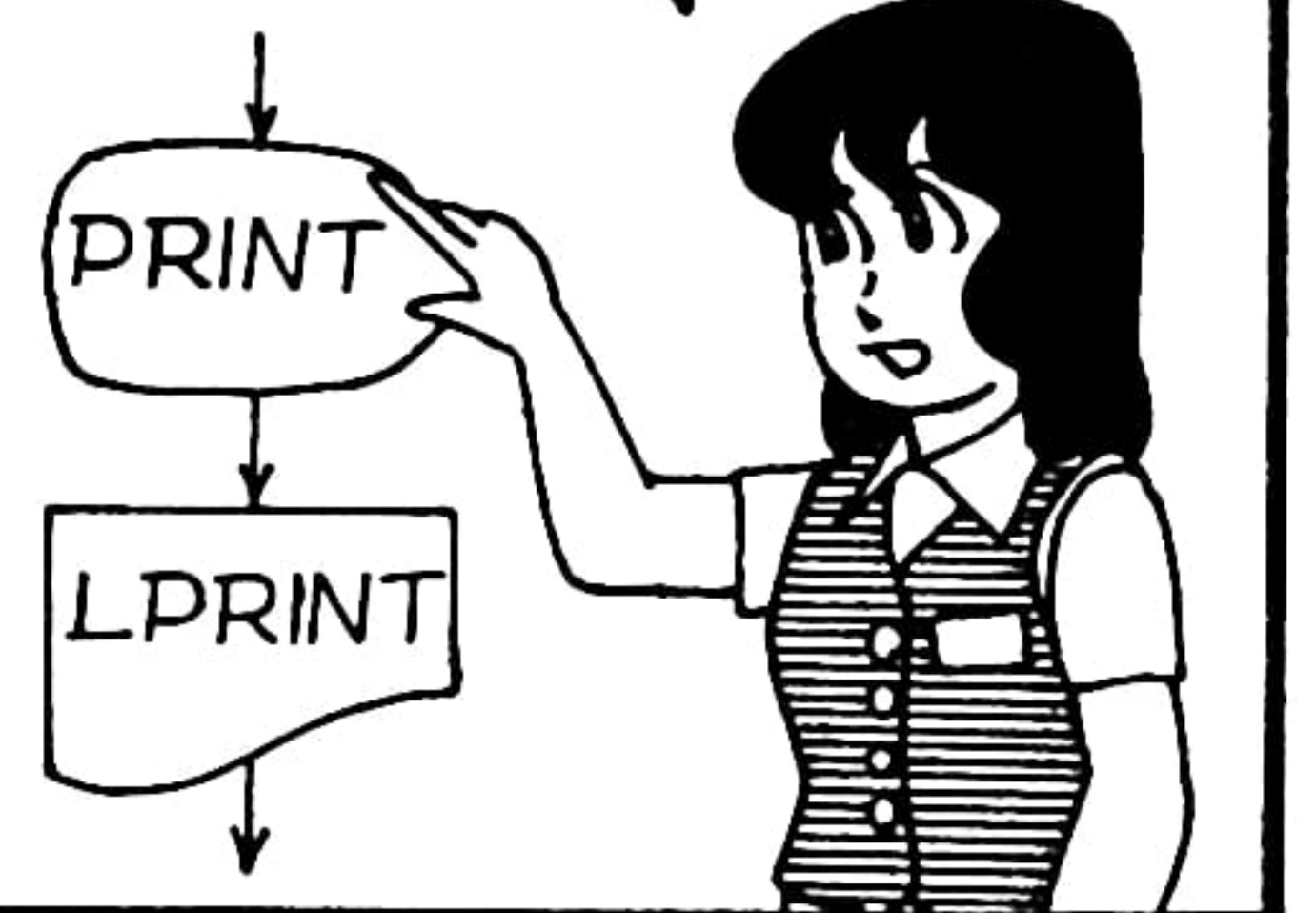
1:ディスクドライブ

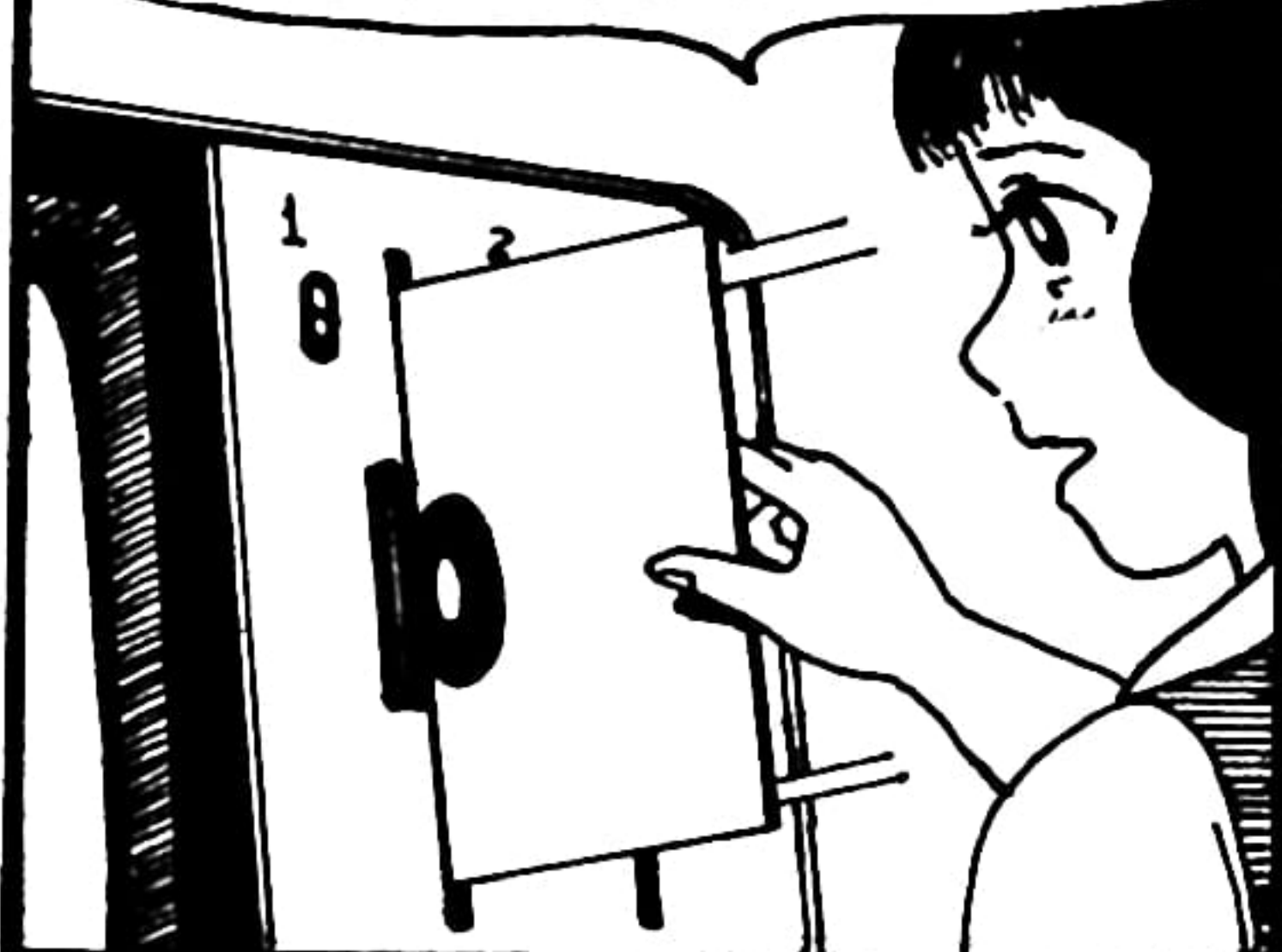
バッファ #1

```
PRINT␣"NAME(L)=";P$:LPRINT␣"NAME(L)=";P$
PRINT␣"NAME(R)=";Q$:LPRINT␣"NAME(R)=";Q$
PRINT␣"SUUJI=";R:LPRINT␣"SUUJI=";R
PRINT␣"DATASUU␣N=";N:LPRINT␣"DATASUU␣N=";N
FOR␣I=1␣TO␣N
PRINT␣USING␣"S(###)=####.##";I;S(I)
LPRINT␣USING␣"S(###)=####.##";I;S(I)
NEXT␣I
```

ポケコンの場合とUSINGフォーマット指定が違っています。

あとはPRINT文で表示すればうまくいったかどうかわかります。ついでですからLPRINT（エルプリント）文で印字もします。

PRINT

LPRINT

では、やってみましょう。まずFDD（フロッピーディスク）をドライブ1に入れます。CP／Mではドライブ1はAと表現されます。2はBです。

1 2

あとは各ステートメントにNo.をつければコーディングの完了です。No.の次に'のあるのは注釈行で、プログラムの実行には関係ありません。

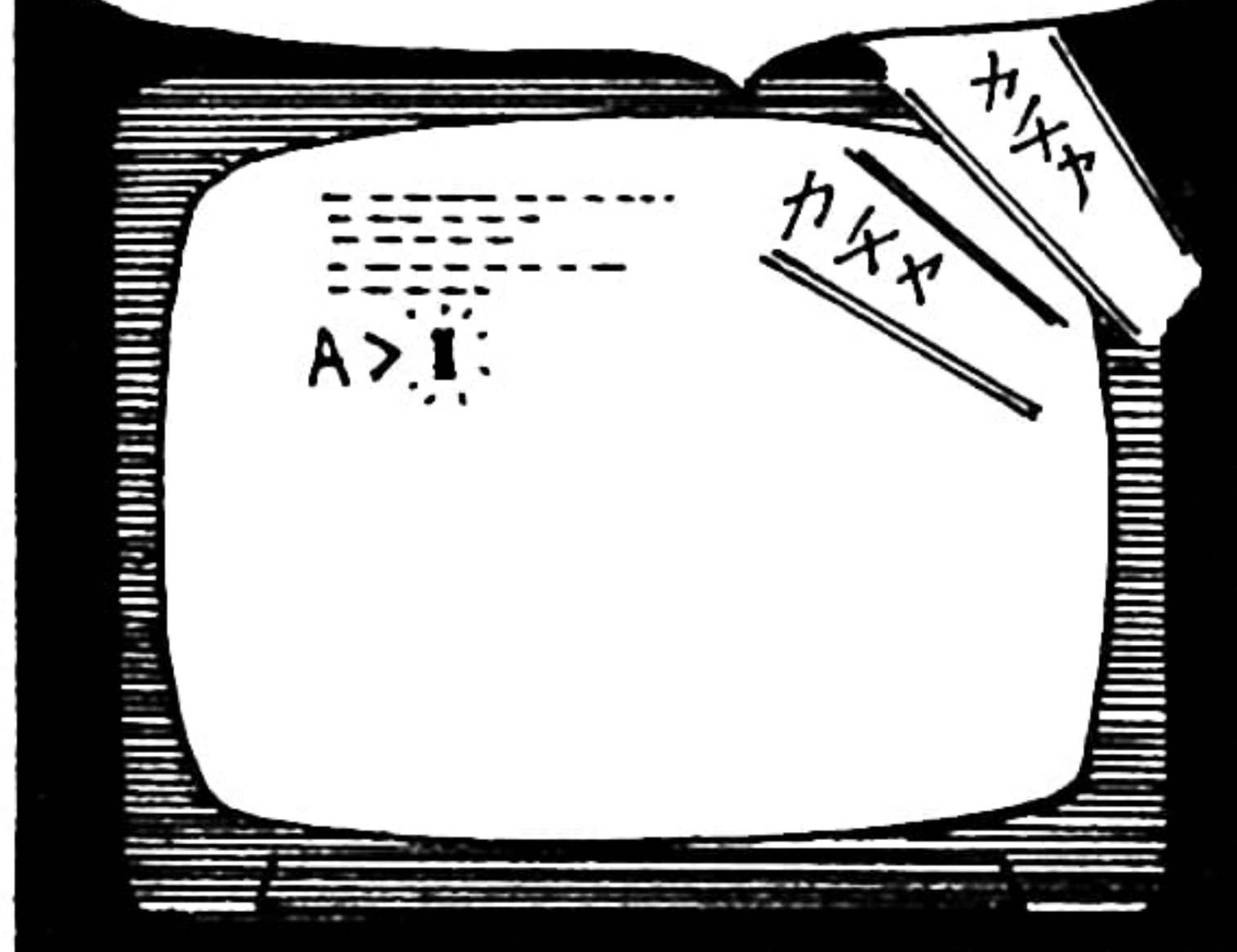
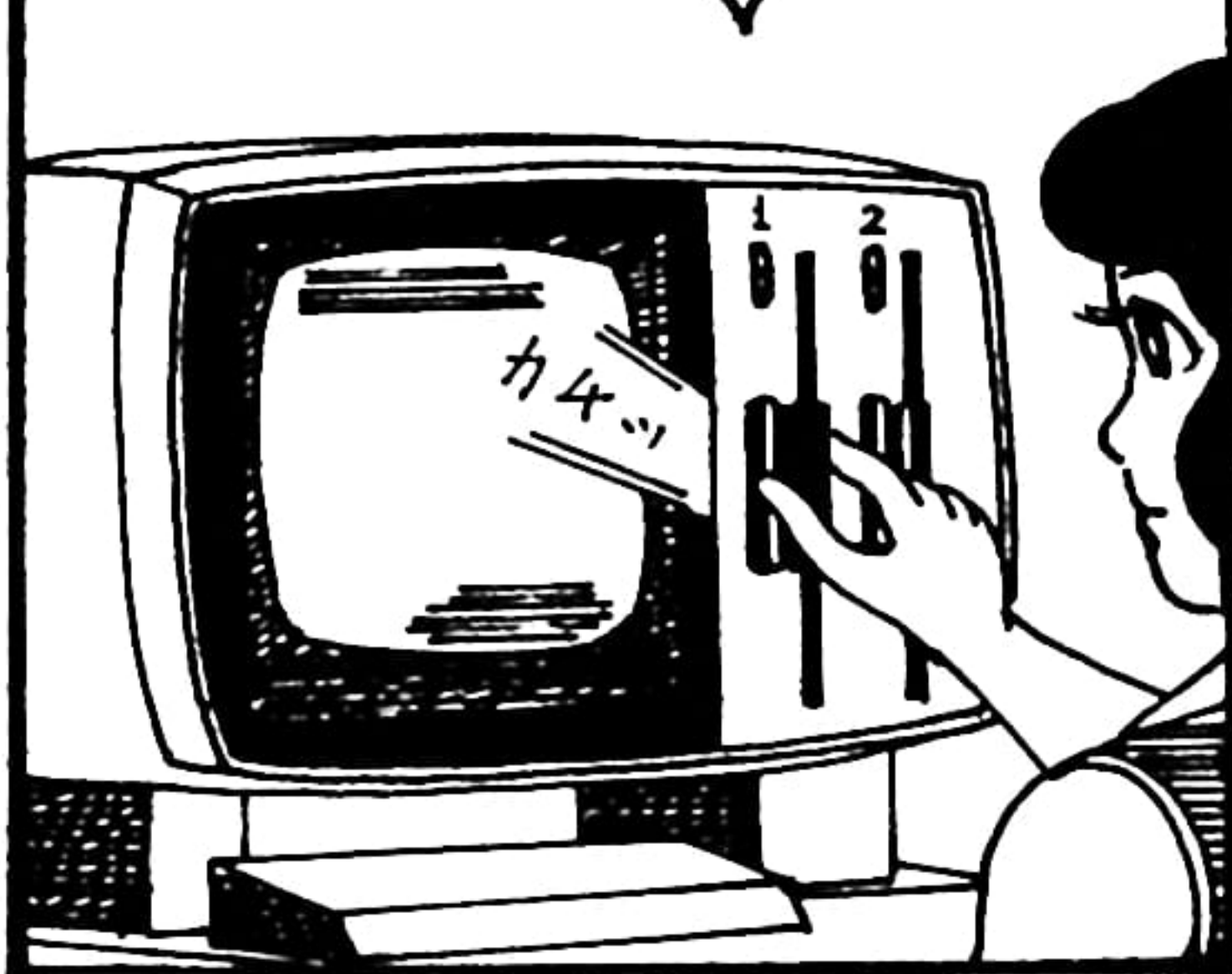
```
10␣'␣SAVE␣CO
20␣'␣AN␣EXAMPLE␣OF
30␣DIM␣S(32):'␣N␣NOSAI
40␣OPEN␣"1:ETUKO"␣AS␣#
```

この場合

このブランクコードもキーインしないと文法エラーになります。

これで終わりですからファイルをクローズします。そして、ENDです。

```
CLOSE␣#1
END
```

これでCP/Mが立ち上がります。表示のA>はCP/Mのプロンプト状態を示し、Aはドライブ1のモードを示しています。
A>
カチャ
カチャ

画面にグリーンの横線が表示されたら最後まで挿入し、フタを閉じます。
カチャッ

フロッピーは途中まで入れ、そのままで電源スイッチをONにします。
パチ

まず、最初の文番号として、10␣が表示されます。ステートメントを入力し RETURN するたび10ずつ増えます。

OK
AUTO
10␣'␣AN␣EXAMPLE␣OF␣
20␣'␣フロッピー␣SAVE␣コード
30␣CLS

AUTOの場合、最初のブランクは自動的にとられます。

次は、プログラムのインプットですが文番号発生機能のAUTOコマンドを使うと便利です。

OK
AUTO

A U T O RETURN

次にOBASIC RETURNでBASICのインタプリタが立ち上がります。このBASICはOSであるCP/Mのもとで走ることになります。プロンプト状態はOKで示されます。

OK
カチャ
カチャ

O B A S I C RETURN

AUTOを解除したい時はCANキャンセルキーを使います。

580␣END
590␣

CAN
カチャ

プログラム上よく使うINPUTやPRINTはCOMDキーとIやPの同時使用で一度に入力されるようになっています。たとえばINPUTならCOMDとIを同時に押せばいいのです。

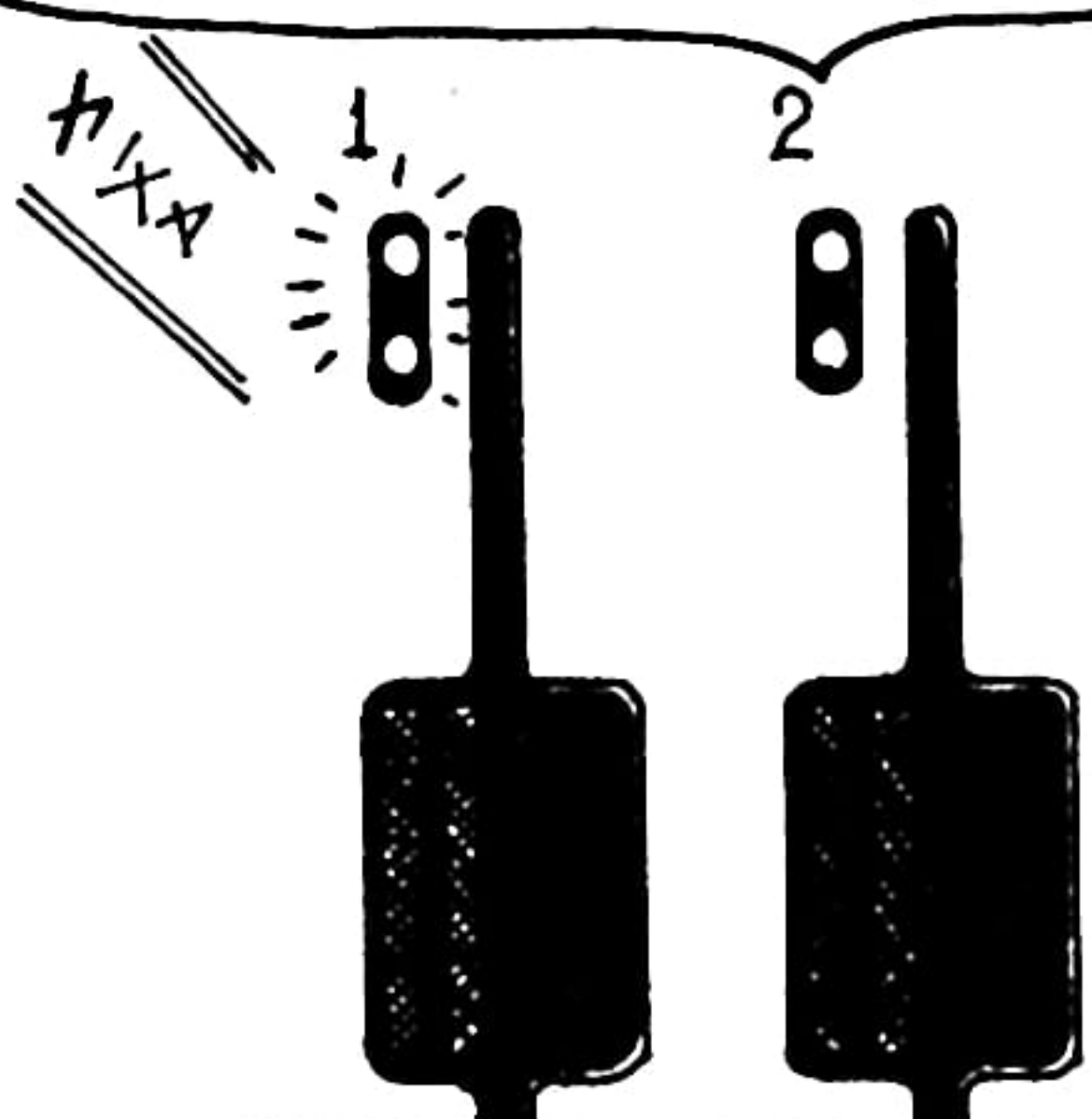
40␣INPUT

COMD カチャ
I カチャ

文番号までいちいちキーインするとスペースの␣まで必要ですし、なかなかめんどうです。
カチャ
カチャ!!

次にINPUT文がありますので、'NAME(L)=' で示されるガイダンスと共に?が表示されます。

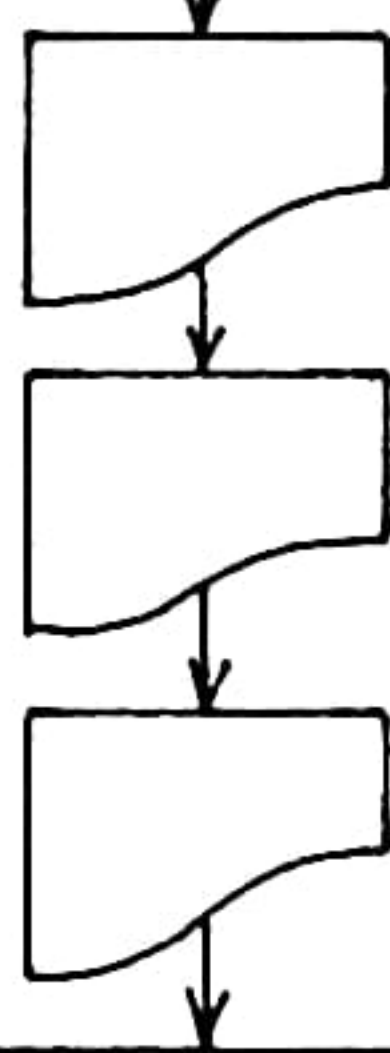
NAME(L)=?

このプログラムには最初にOPEN文がありますので、ドライブ1が起動します。
カチャ
1
2

プログラムの実行はRUNコマンドです。ファンクションキーにRUNと同じ機能のものがあるはずですが、もちろん、ひとつひとつキーを押していってもいいのです。

RUN

カチャ カチャ カチャ
R U N RETURN

表示と印字は、インプット内容の確認、クリアの確認、ファイルからの読出し確認のため３回することにしました。

表示された項目のデータを入れていきます。

NAME(L)=?␣TAKUTO タクト
NAME(R)=?␣MASAYOSI マサヨシ
SUUJI(####.##)=?␣123.45
DATASUU␣N(Max.32)=?␣3
DATA(####.##)=?␣111.11
DATA(####.##)=?␣222.22
DATA(####.##)=?␣333.33

この例ではN＝３なので３つ目のデータを入れ終わった時にプログラムが次に進みます。

. 2 2 RETURN
. 3 3 RETURN

次の360番のGOSUB文ではクリアされた内容がプリントされます。Nはプログラム上クリアできません。

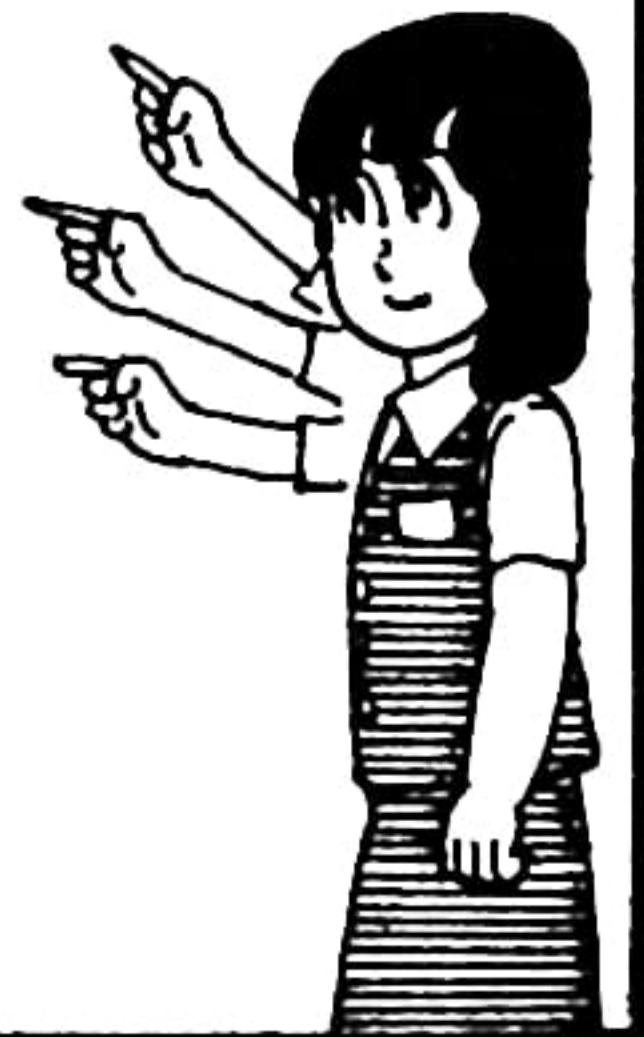
DATA(␣␣3)= 333.33

NAME(L)=
NAME(R)=
SUUJI=0
DATASUU N= 3
DATA(␣␣1)= 0.00
DATA(␣␣2)= 0.00
DATA(␣␣3)= 0.00

最初の135番のGOSUB文ではインプットの内容がそのまま出力されます。

NAME(L)= TAKUTO タクト
NAME(R)= MASAYOSI マサヨシ
SUUJI=123.45
DATASUU N= 3
DATA(␣␣1)= 111.11
DATA(␣␣2)= 222.22
DATA(␣␣3)= 333.33

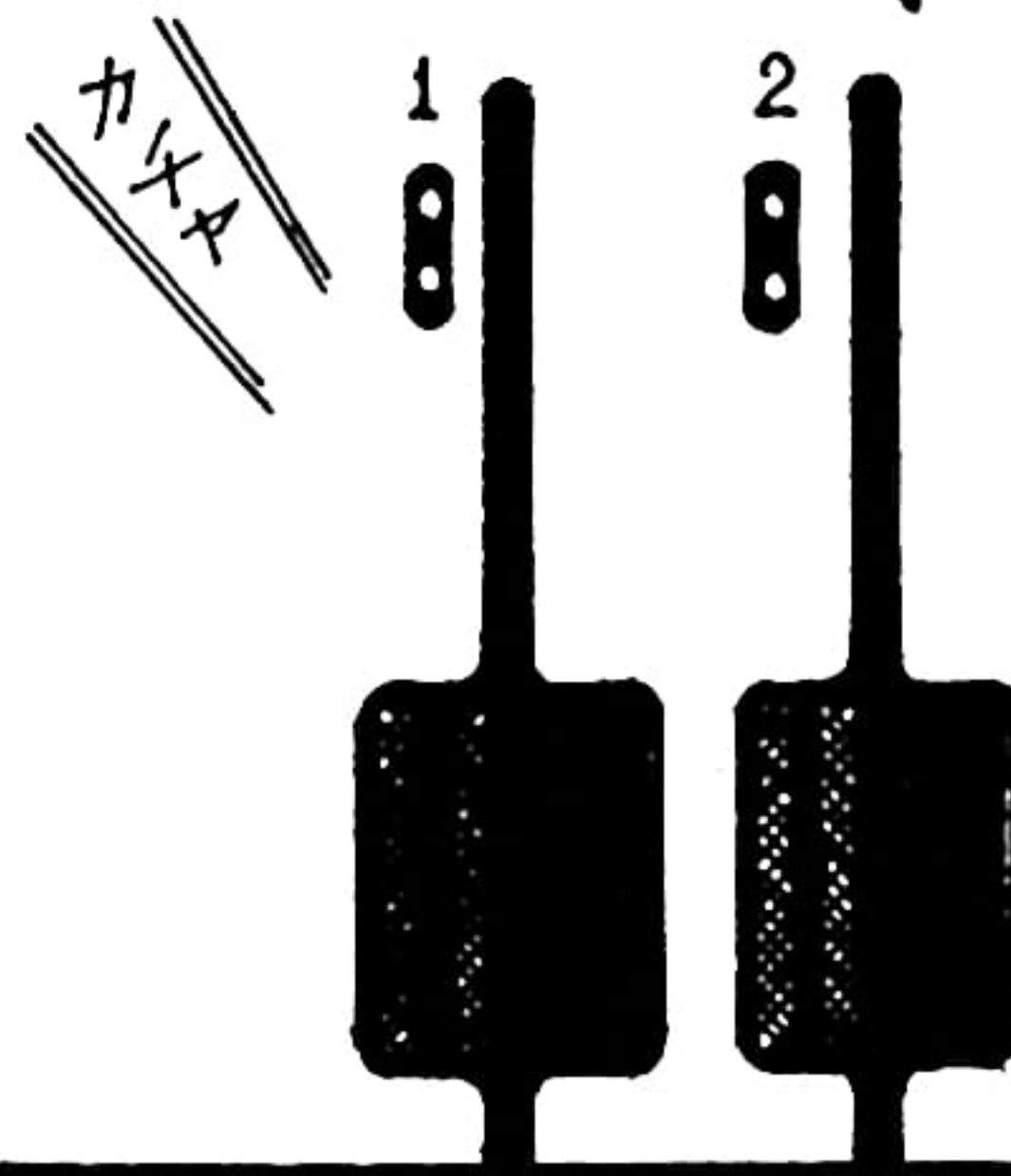
同じ変数の内容確認なので、表示、印字ルーチンをサブルーチンにしました。

135␣GOSUB␣490
:
360␣GOSUB␣490
:
470␣GOSUB␣490
:
GOTO 570
490␣PRINT␣
:
570␣RETURN
END

これでなんとなくランダムファイルが使える気がしてきました。自由自在に使いこなせるのは時間の問題です。

んよしよし

そして、ファイルをCLOSEし、END文でプログラム終了です。

カチャ

1
2

最後に470番でファイルから読んだ内容を出力します。

DATA(␣␣1)= 0.00
DATA(␣␣2)= 0.00
DATA(␣␣3)= 0.00

NAME(L)= TAKUTO タクト
NAME(R)= ␣␣ ␣␣MASAYOSI マサヨシ
SUUJI= 123.45
DATASUU N= 3
DATA(␣␣1)= 111.11
DATA(␣␣2)= 222.22
DATA(␣␣3)= 333.33

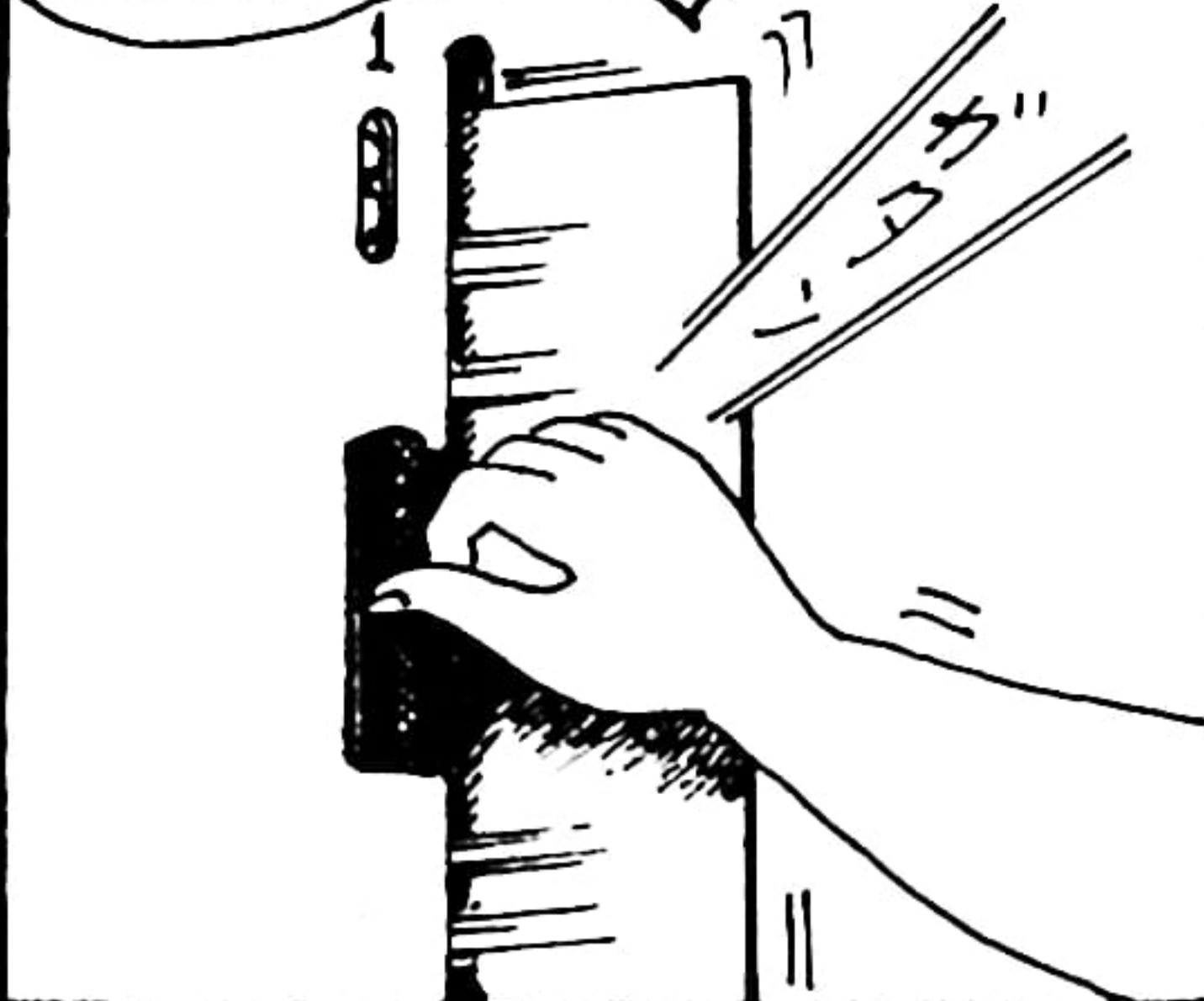
ここにRとLの差が出ています。

最初ETUKO 1でSAVEしたのですがあれこれ修正しているうちに改訂９になりました。

ETUKO1
ETUKO2
:
ETUKO8
ETUKO9

FDはRESET RETURN と入れすべてのファイルをCLOSEしてから取り出してください。

1

ガチャ

プログラムは一応FDDにしまっておきましょう。

SAVE␣"ETUKO9"

カチャ

プログラムの内容を変更したい時は名前も変えてSAVEした方がいいでしょう。

フロッピーの中のファイルを確認してみましょう。

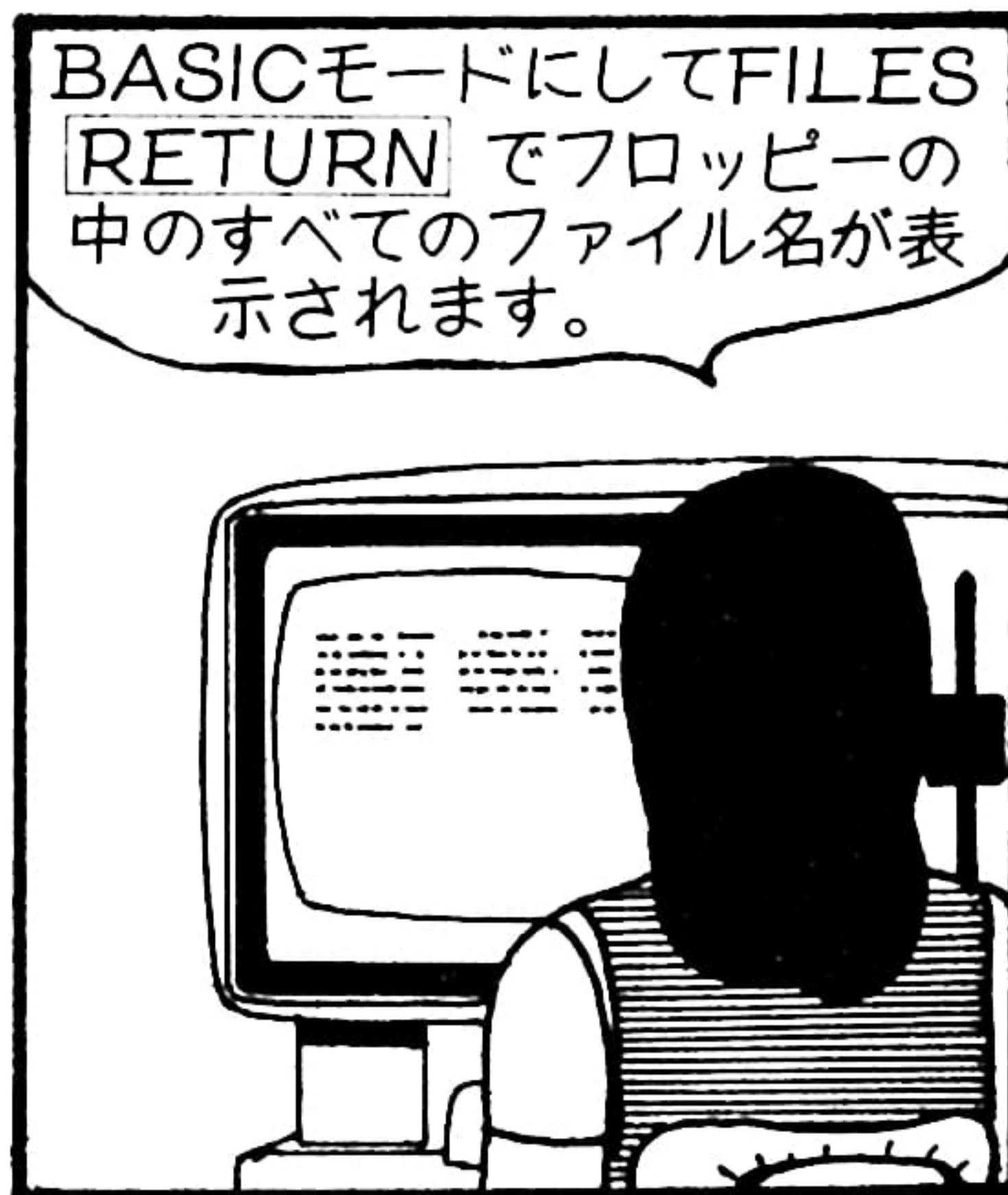
BASICモードにしてFILES RETURN でフロッピーの中のすべてのファイル名が表示されます。

LFILES RETURN でそれらのファイル名がプリントされます。

ビィーッ ビィーッ カチャ

ランダムファイルで使ったETUKOもありました。

ETHUKO,ETUKO1～ETUKO9は、テストプログラムにつけた名前です。

その他のファイル名は、最初から入っているシステムプログラムです。

```
MOVCPM  .COM   PIP     .COM   SUBMIT  .COM   XSUB    .COM   ED      .COM
ASM     .COM   DDT     .COM   LOAD    .COM   STAT    .COM   SYSGEN  .COM
DUMP    .COM   DUMP    .ASM   BIOS    .ASM   CBIOS   .ASM   DEBLOCK .ASM
DISKDEF .LIB   FDDUTY  .COM   KANJI   .COM   IF20T030.DOC   IFDEM01 .BAS
OBASIC  .COM   IFINSTL .COM   IFCONFIG.COM   2       .      A       .
ETUKO   .      ETHUKO  .      ETUKO1  .      ETUKO2  .      ETUKO3  .
ETUKO4  .      ETUKO5  .      ETUKO6  .      ETUKO7  .      ETUKO8  .
ETUKO9  .
```

こういう名前のファイルがこのフロッピーの中に入っている。

これはN＝5の場合のサンプルリストです。

```
NAME(L)=TAKUTO ﾀｸﾄ
NAME(R)=MASAYOSI ﾏｻﾖｼ
SUUJI= 123.45
DATASUU N= 5
DATA(  1)=  111.11
DATA(  2)=  222.22
DATA(  3)=  333.33
DATA(  4)=  444.44
DATA(  5)=  555.55
```

KEYINした内容を印字したもの

```
NAME(L)=
NAME(R)=
SUUJI= 0
DATASUU N= 5
DATA(  1)=    0.00
DATA(  2)=    0.00
DATA(  3)=    0.00
DATA(  4)=    0.00
DATA(  5)=    0.00
```

変数をクリアした後印字したもの

```
NAME(L)=TAKUTO ﾀｸﾄ
NAME(R)=                         MASAYOSI ﾏｻﾖｼ
SUUJI= 123.45
DATASUU N= 5
DATA(  1)=  111.11
DATA(  2)=  222.22
DATA(  3)=  333.33
DATA(  4)=  444.44
DATA(  5)=  555.55
```

フロッピーから読み出した内容を印字したもの

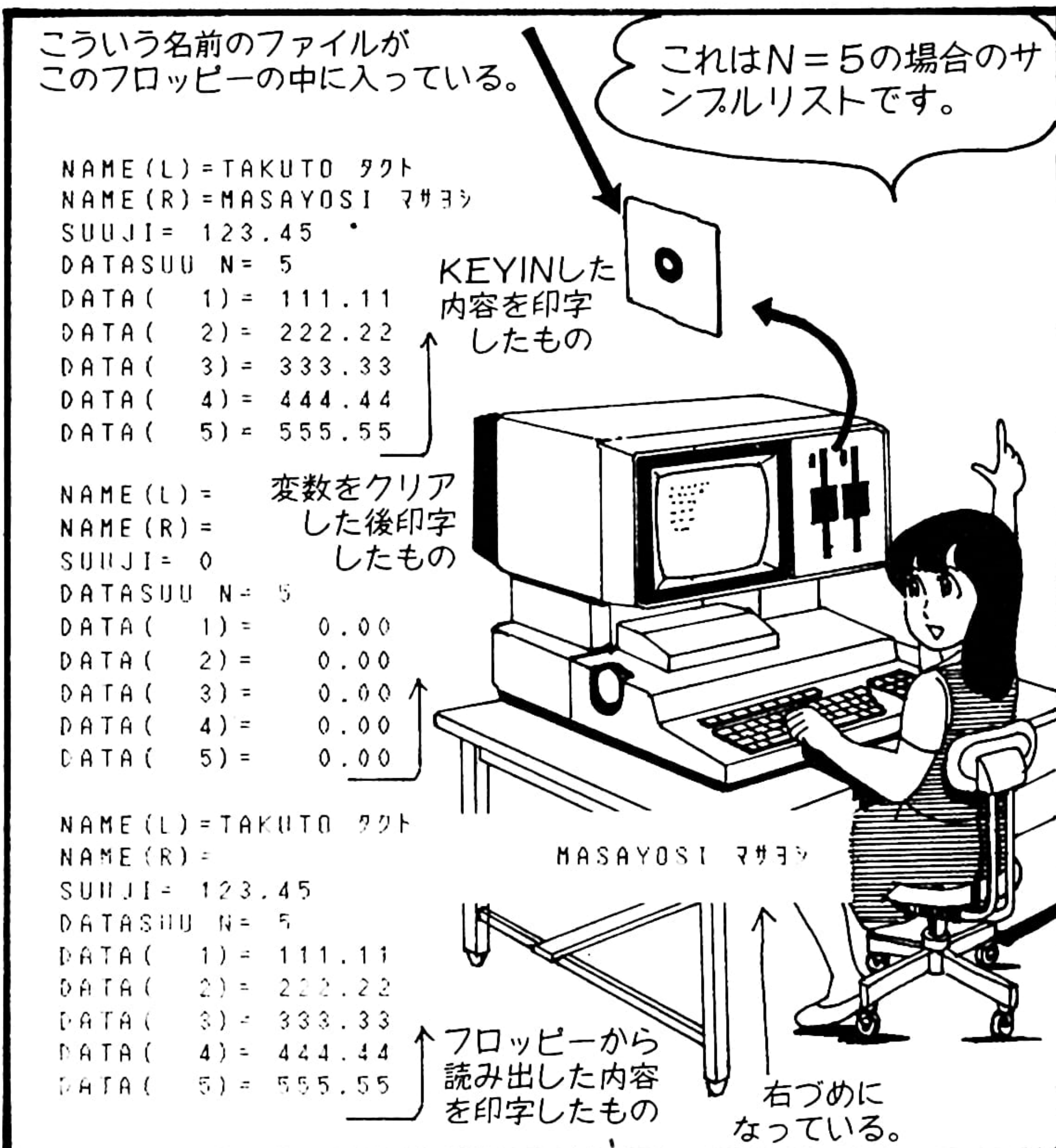
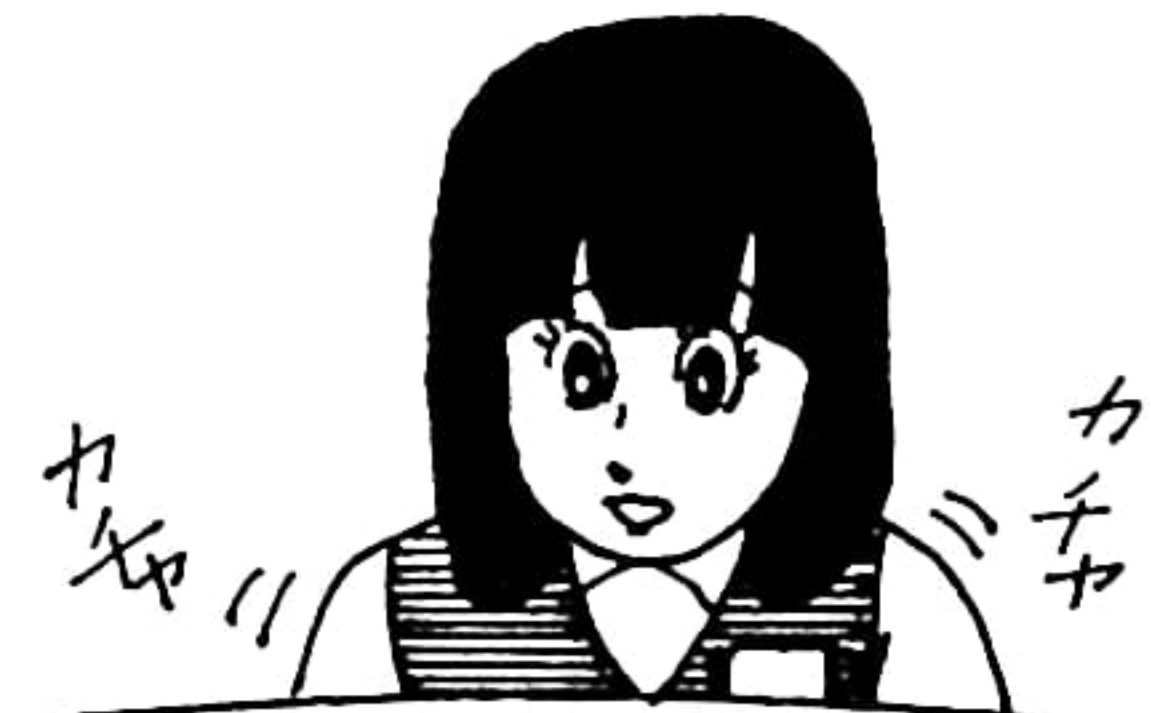
右づめになっている。

ファイル名ETUKO9のプログラムをメインメモリにロードしたい時は、LOADコマンドを使います。

LOAD "ETUKO9"
OK

カチャ カチャ

ロードが完了したらOKが表示されるのでRUN RETURN でプログラムを走らせます。

カチャ カチャ

```
10 'AN EXAMPLE OF RANDAM FILES HANDLING
20 ' ﾌﾛｯﾋﾟｰ SAVE ｺｰﾄﾞ ﾊ ETUKO9
30 CLS
40 DIM S(32)
50 OPEN "1:ETUKO" AS #1
60 FIELD #1,20 AS A$,30 AS B$,10 AS C$,128 AS D$
70 INPUT "NAME(L)=";P$:'ﾋﾀﾞﾘﾂﾞﾒ ﾓｼﾞ
80 INPUT "NAME(R)=";Q$:'ﾐｷﾞﾂﾞﾒ ﾓｼﾞ
90 INPUT "SUUJI(####.##)=";R:'ｽｳｼﾞ
100 INPUT "DATASUU N(Max.32)=";N
110 FOR I=1 TO N
120 INPUT "DATA(####.##)=";S(I)
130 NEXT I
135 GOSUB 490:PRINT:LPRINT
140 R$=MKS$(R)
150 FOR I=1 TO N
160 S$=S$+LEFT$(MKS$(S(I)),4):' 4ﾊﾞｲﾄ ｹﾂｺﾞｳ
170 NEXT I
180 LSET A$=P$:' ﾊﾞｯﾌｧ ﾍﾉ ｾｯﾄ
190 RSET B$=Q$
200 LSET C$=R$
210 LSET D$=S$
220 PUT 1,2:' ﾊﾞｯﾌｧ #1 ｶﾗ ﾌﾛｯﾋﾟｰ ﾉ
230 '         ETUKO ﾀﾞｲ 2 ｾｸﾀ ﾆ ｶｷｺﾑ
250 P$="":' ﾘｾｯﾄ ﾙｰﾁﾝ
260 Q$=""
270 R=0:R$=MKS$(R)
280 S$=""
290 FOR I=1 TO N
300 S(I)=0
310 NEXT I
320 LSET A$=P$
330 RSET B$=Q$
340 LSET C$=R$
350 LSET D$=S$
360 GOSUB 490:PRINT:LPRINT:' ﾌﾟﾘﾝﾄ (ﾘｾｯﾄ ｶｸﾆﾝ)
370 GET 1,2:' ETUKO ﾀﾞｲ 2 ｾｸﾀ ｶﾗ
380 '         ﾊﾞｯﾌｧ #1 ﾆ ｶｷｺﾑ
390 P$=A$
400 Q$=B$
410 R$=C$:R=CVS(R$)
420 S$=D$
430 FOR I=1 TO N
440 I1=1+4*(I-1)
450 S(I)=CVS(MID$(S$,I1,4)):' 4ﾊﾞｲﾄ ﾆ ﾌﾞﾝｶｲ
460 NEXT I
470 GOSUB 490:GOTO 580:' ﾌﾟﾘﾝﾄ (ﾖﾐﾀﾞｼ ｶｸﾆﾝ)
480 ' ﾌﾟﾘﾝﾄ ﾙｰﾁﾝ
490 PRINT "NAME(L)=";P$:LPRINT "NAME(L)=";P$
500 PRINT "NAME(R)=";Q$:LPRINT "NAME(R)=";Q$
510 PRINT "SUUJI=";R:LPRINT "SUUJI=";R
520 PRINT "DATASUU N=";N:LPRINT "DATASUU N=";N
530 FOR I=1 TO N
540 PRINT USING "DATA(###)=####.##   ";I;S(I)
550 LPRINT USING "DATA(###)=####.##   ";I;S(I)
560 NEXT I
570 RETURN
580 CLOSE #1:END
```

データのキーイン

フロッピーへの書き込み

ETUKO 9のプログラムリストです。

変数のクリア

フロッピーからの読み出し

ソフトメカニズムの表現にチャレンジしてみました。

この　の中にファイル名を書きます。

ドライブ
ドライブ
"1:[ ]"
"2:[ ]"
"PAC1: [ ]"
PRINTER
ROM
"LPT1:"
RS-232C 通信ポート
バッファ
"COM1:"
ビデオRAM
"SCRN:"
キーボード
"KYBD:"
PUT文
GET文
#1
256バイト バッファレジスタ
変数参照表
FIELD文
FIELD文
⋮
CVS関数 LEFT$ RIGHT$
LSET文 RSET文
MKS$関数
OPEN文 CLOSE文
CPU
FIELD文
メインメモリ

OS / BASIC インタプリタ / プログラムエリア / データエリア / I/O

OS用コマンド
BASIC RETURN

BASIC用コマンド
AUTO RETURN
SAVE
LOAD
DELETE
NEW
RUN
LIST
LLIST

ステートメント ファンクション
OPEN
FIELD
LSET
RSET
MKS$
PUT
GET
CVS
RIGHT$
LEFT$
MID$
CLOSE

変数名
P$
Q$
R
S(I)

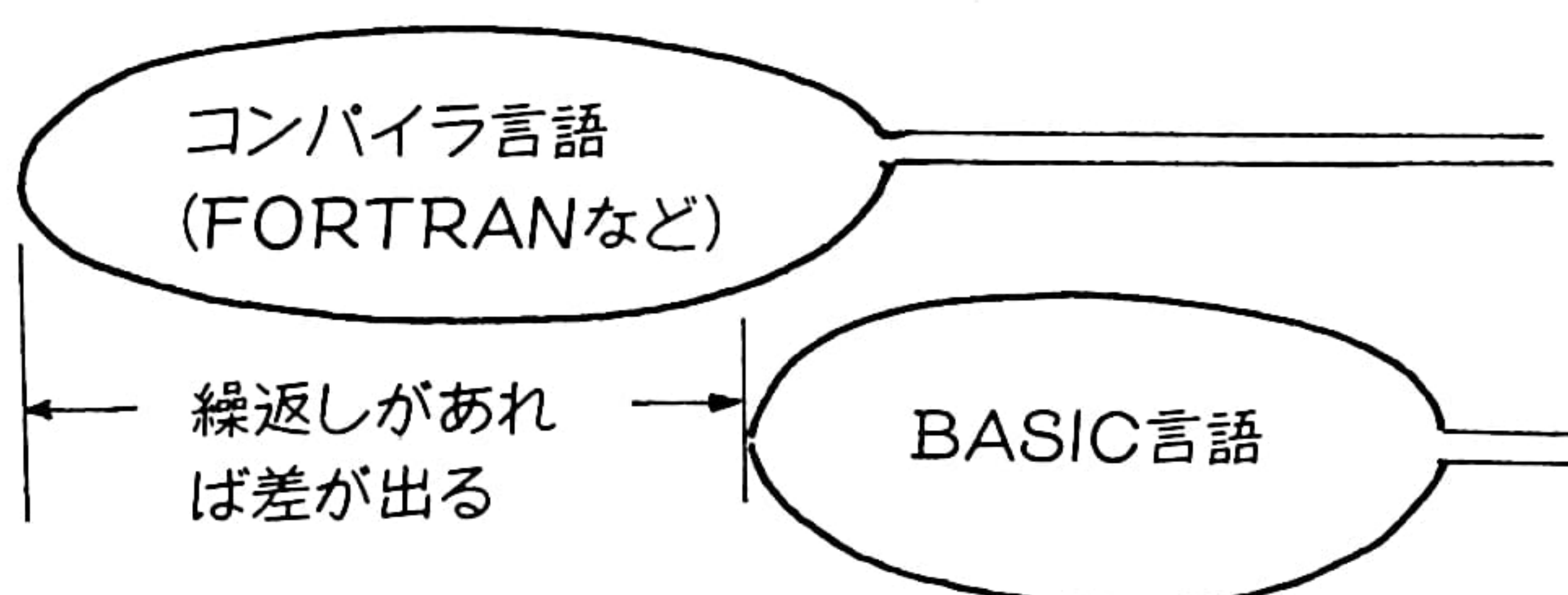
メーカーや型式が違うと多少表現方法が変わります。

BASIC Q&A

＊BASICは処理スピードが遅い？

使う処理の対象によっては、確かにコンパイル型言語によるプログラム処理よりも遅く、気になる場合があります。たとえば低速I/O回数の少ないグラフィック処理、それに解析用技術計算などです。

しかし、最近の各メーカーの提供するBASICは、それぞれ工夫を加え、グラフィックスやI/O命令も充実させてきており、十分実務に活用できる環境をつくりあげつつあります。しかし、それを使いこなし、活用する人材の層がまだまだ薄いのが現状です。

コンパイラ言語
(FORTRANなど)
繰返しがあれば差が出る
BASIC言語

＊BASICは互換性がないのでは？

高級言語であるBASICは、どのパソコンでも走らなければならないところですが、現実にはA社のマニュアルで作ったプログラムがそのままでB社のパソコンで走るということはありません。しかし、同じBASICなのですからまるっきり違うということはなく、少し修正すれば走るのが普通です。とくにI/O関係の命令がまちまちです。

RS-232Cファイルオープンもこう違う

A社	B社
OPEN␣"COM1:4N81" AS␣1 (ひとつの文で入出力宣言)	OPEN␣"COM:4N81NN"␣ FOR␣INPUT␣AS␣#1 OPEN␣"COM:"␣ FOR␣OUTPUT␣AS#1 (入力と出力を別々に宣言)

＊PASCAL(パスカル)やC(シー)が注目されているようですが？

両方共コンパイル型言語です。PASCALは、はっきり言って教授が学生のプログラムを採点しやすいように作ったもの、という印象です。たとえば印字のための命令すらなく、ただCRTに結果を表示できるだけです。それをしようとすればそのマイコンのマシン語が使える能力がなければダメなのです。CはPASCALよりもさらにアセンブラに近い低級言語です。これも標準ではCRTに表示するだけですから、実用的に活用するためには高度の知識が必要です。これらのソフトは10数万で買えますが、ディスク付のマイコンシステムでCP/Mが使える環境が必要なため、誰でも簡単にトライできるというものではありません。

B マシン語がわかればもっと使いこなせる

CPUの系図(言葉の違いは育ちの違い)

インテル家

電卓競争時代

ハードの共通利用

i4040

8ビットCPU 8080

8085(8080とソフト的にはコンパチ)ハード面を簡素にしたにとどまった

16ビットCPU 8086

16ビットCPU

モトローラ家

ミニコンのOS UNIXの考え方を盛り込む形でつくられた。

8808

8ビットCPU 6800

6809

16ビットCPU 68000

モステクノロジー家

パソコン用に開発された。(APPLEなど)

8ビットCPU 6502

ザイログ家

Z80(8080命令コードをそのまま生かし、ハードを簡潔にし、ニーモニックを整備し、大幅に命令コードを追加した)

16ビットZ8000 8080のコード体系をやめた

ビジネス用パソコンは16ビット時代に入った。

(基本ソフト)ディジタルリサーチ家

CP/M-80(ソフト)

CP/M-86(ソフト)

マイクロソフト家

MS-DOS(ソフト)

32ビットCPUへ

図形処理パソコンなど32ビットCPUを各社で開発中である

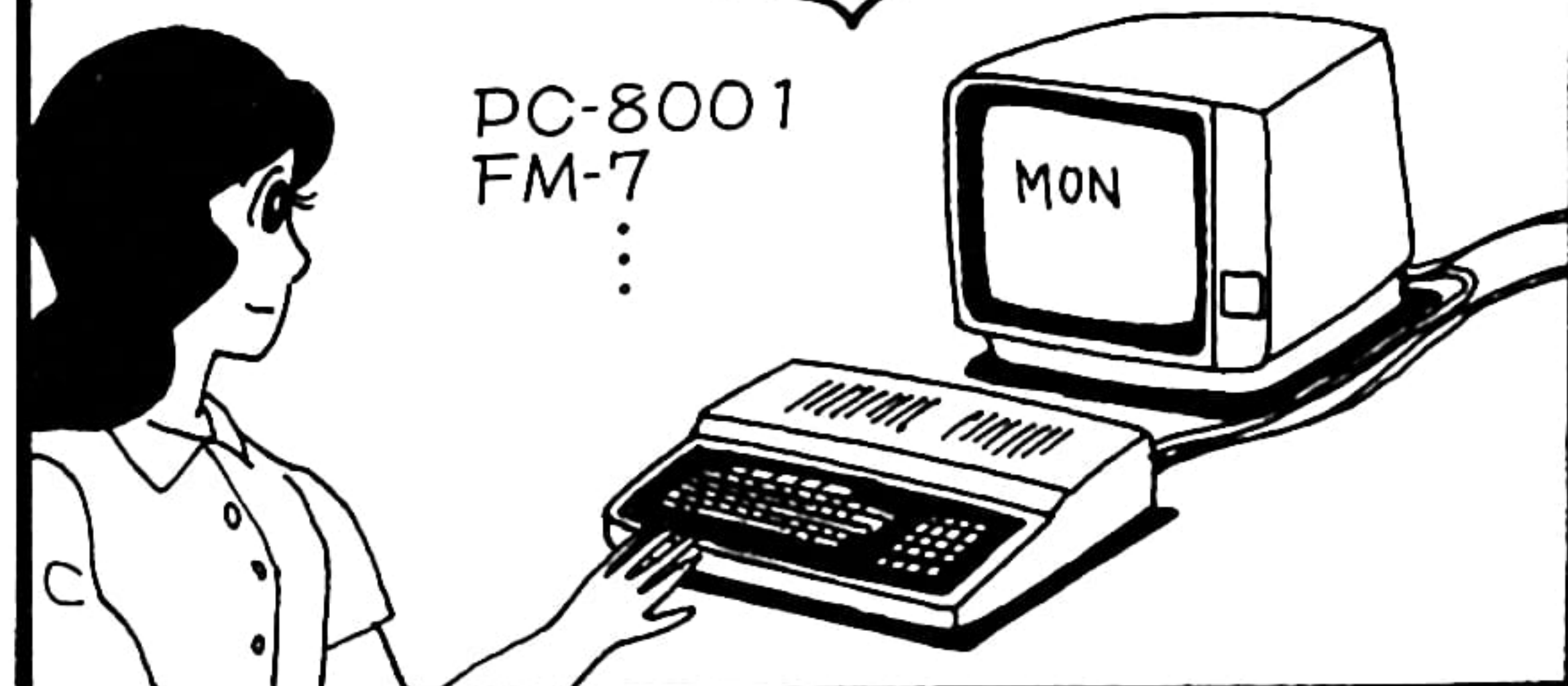
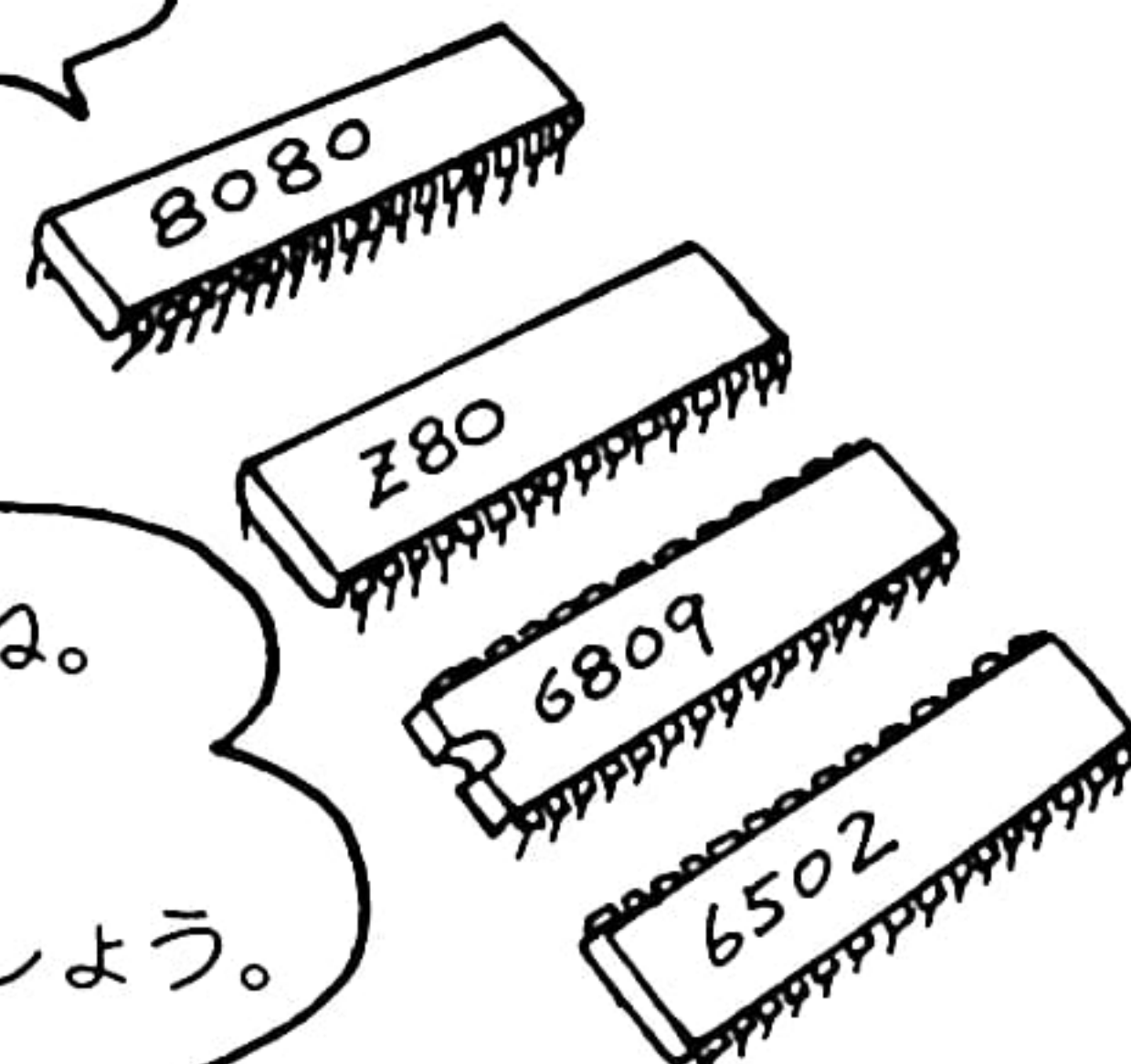
①マシン語との出会い

パソコン（ROM型）ではMONコマンドでマシン語らしきものに出会うことができます。

PC-8001
FM-7
⋮

MON

シーン

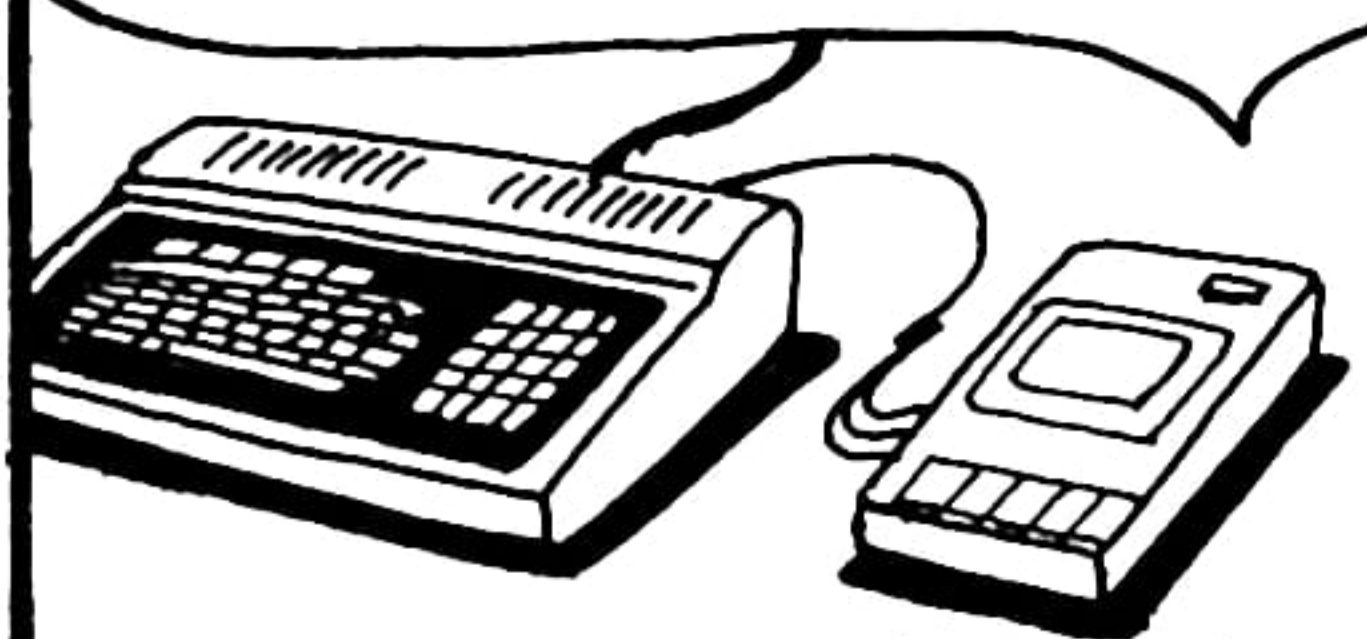
8080
Z80
6809
6502

ね、ね。お話しましょう。

ま、とにかくMON RETURN とKEYをたたいてみましょう。すると＊印が現れました。

NEC PC-8001 BACICver
Copyright 1979 by Micros
OK ←── 電源ONでここまで表示
MON ← KEYインした内容
＊

モニタって何？　ということになりますが、それはこれからお話する、ワンボードマイコンを使って説明します。

このMONは（モンスタ）のモンではなく（モニタ）の略語です。

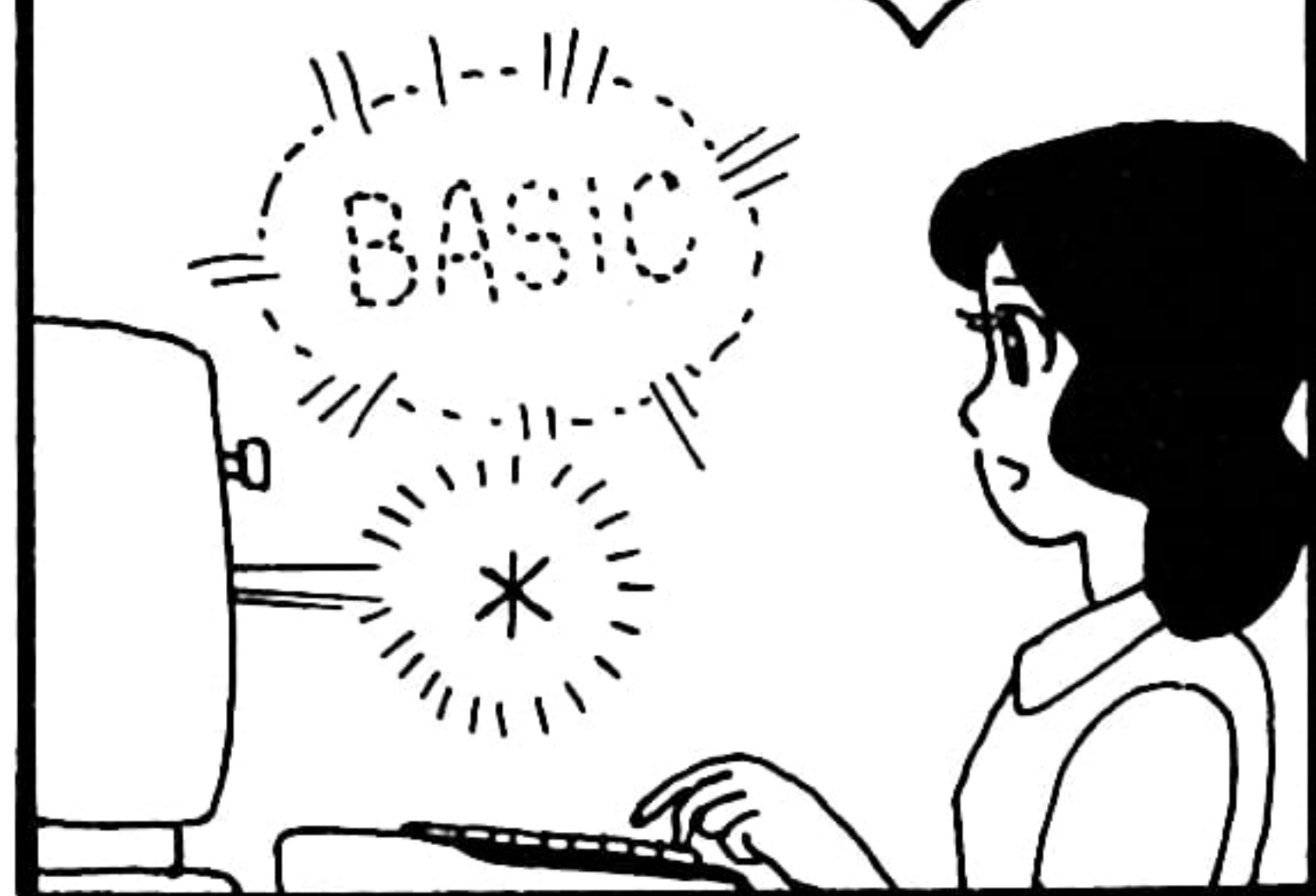
MONITOR
MON
RETURN

たとえばインベーダーなどのマシン語のテープを読み込ませたい時は L RETURN とします。うまく読み込みが完了したらまた＊表示になります。

カセットを入れ再生状態にしておく。

PC-8001のモニタのコマンド
- S…RAMにデータを書き込む
- D…メモリの内容を表示する
- L…テープからマシン語を入れる
- LV…ロードベリファイ
- W…テープにマシン語を録音
- G…プログラムの実行
- CTRL B…BASICに戻る
- TM…テストメモリ

（FM-7/8は別のコマンド）

これが、モニタのコマンド待ちの状態です。BASIC言語は使えなくなりました。

BASIC
＊

ここで表示されたC3とか06とかいうのが実はマシン語（に近い）言語なのです。まずワンボードマイコンでお話しましょう。

DC000,C00F RETURN とキーインするとC000番地からC00F番地までのデータが表示されます。

```
＊DC000, C00F
C000 C3065839…
C007 003832FC…
＊
```

↑アドレス　↑数字2つがひとつのデータ

プログラムの実行はG（ゴー）コマンドです。テープの解説に書いてあるアドレスを入れるとプログラムが走ります。
GC000 RETURN

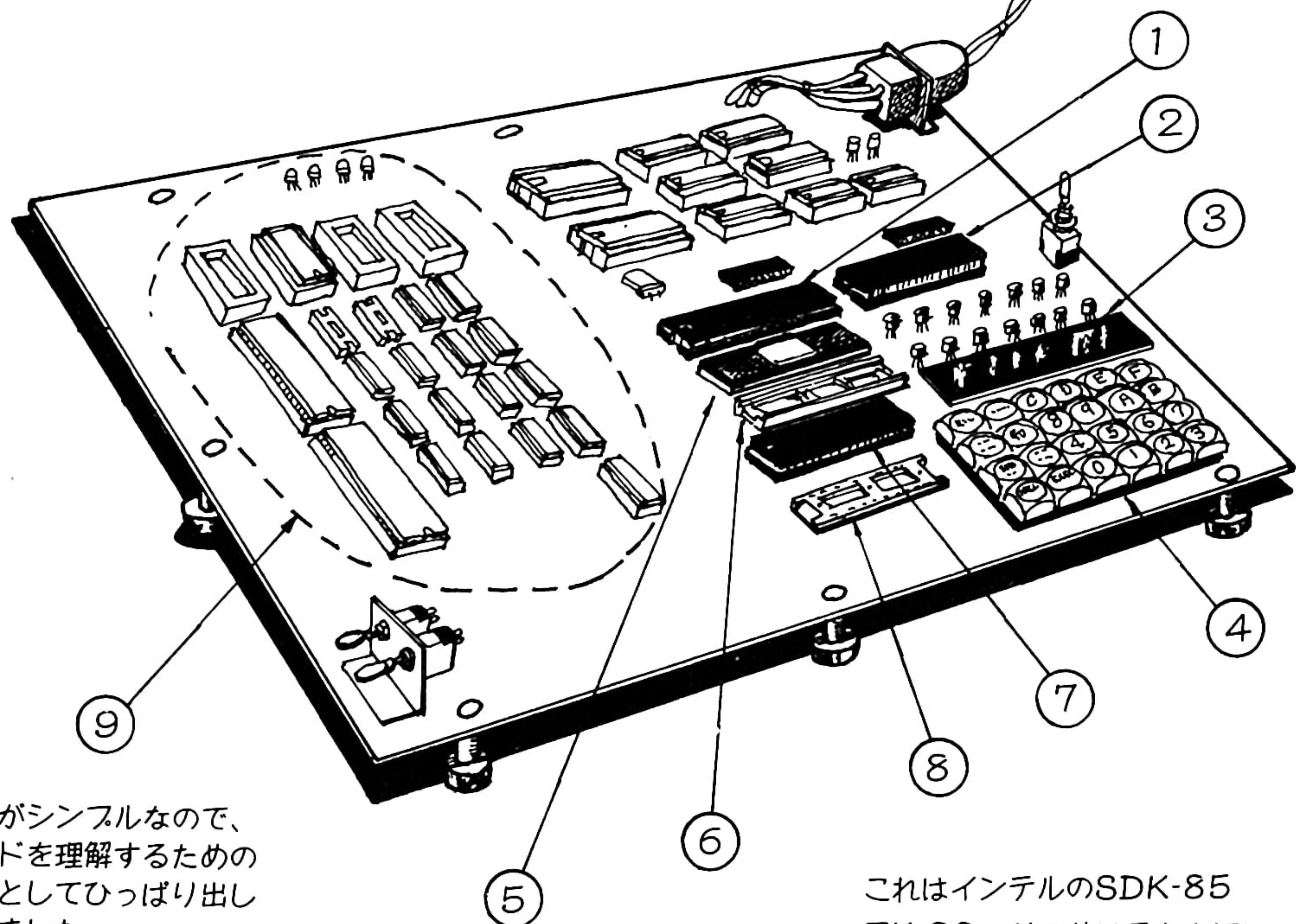
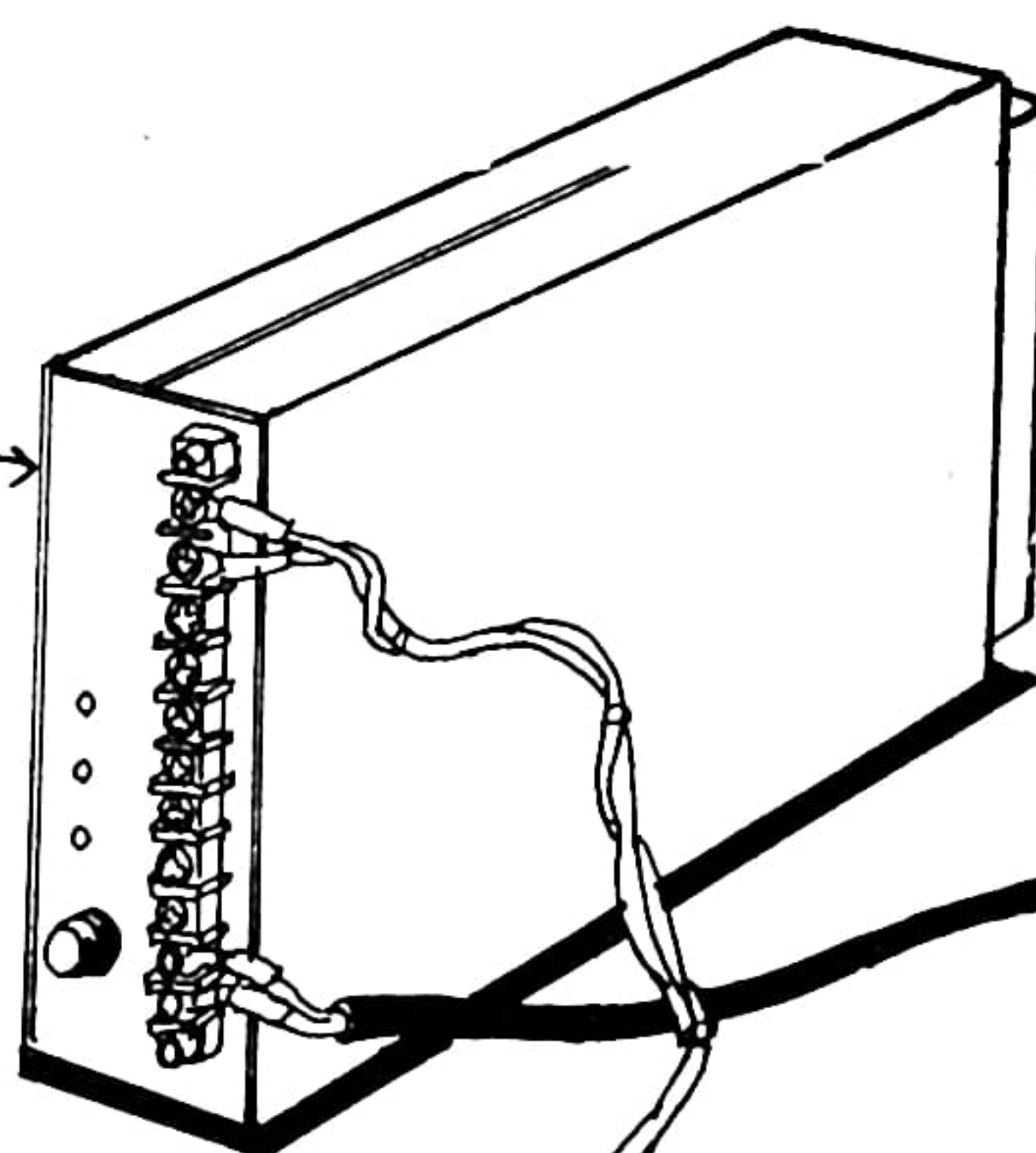
②ワンボードマイコンとモニタ

HELP

CRT なんてないもんね。７セグメントのLED表示

その昔、一世を風靡したワンボードマイコンです。

DC5V AVR→
直流DC 5V
の定電圧装置
ＩＣの駆動用
です。

①
②
③
④
⑨
⑧
⑦
⑥
⑤

構成がシンプルなので、ハードを理解するための教材としてひっぱり出してきました。

これはインテルのSDK-85
TK-80、その他いろんなワンボードマイコンがありました。

① CPU8085（8080とほとんど同じ）
② キーボード・ディスプレイコントローラ8279
③ ７セグメント表示器（アドレス４桁データ２桁）
④ 16進キーボード（プログラムは０～９、Ａ～Ｆのキーを使ってここから入れます）。
⑤ ROMメモリ
⑥ ROMメモリ増設用ソケット
⑦ RAMメモリ
⑧ RAMメモリ増設用ソケット
⑨ オプション用ユニバーサル部分

今は７と表示している。

０～９を入れて 128 通りの表示のしかたがあります。

７セグメント
LED

はじめての人には、このヘキサという数字にも戸惑われるかもしれませんが、わからなくても先に進んでください。そのうちわかります。

HEXA

0、1、2、3
4、5、6、7
8、9、A、B
C、D、E、F

このキットにはすでに固定プログラムが入っていますので、私達は16進数(ヘキサ)でデータを扱うことができます。

モニタプログラム

これが、このキットのインプット部です。プログラム、データは、このキーを使ってインプットします。

RESET	INTR	C	D	E	F
SINGLE STEP	GO	8 H	9 L	A	B
SUBST MEM	EXAM REG	4 SPH	5 SPL	6 PCH	7 PCL
NEXT .	EXEC .	0	1	2	3

プログラムの内容はアドレス表示用4桁に'HELP'を点滅させるものです。

HELP

この表示器を使うプログラムがマニュアルにありますので、これをインプットして走らせてみましょう。

アウトプットは、この6桁のLED(発光ダイオード)表示です。データを扱う時は、アドレス4桁データ2桁で表示されます。

アドレス用　　データ用

8.8.8.8.　8.8.

'HELP'の点滅プログラム
(SDK-85以外のキットには使えません)

この簡単な例でマシン語とCPUの働きを理解してもらおうと思います。

(右のリストのアセンブラプログラムですが、このキットでは使えません。)

アセンブラ　　ニーモニック

```
       LXI SP,20C8H
       MUI A,1
       MVI B,0
       LXI H,2004H
       CALL OUTPT
DPY :  MVI A,0
       MVI B,0
       LXI H,2000H
       CALL OUTPT
       LXI D,0FFFFH
       CALL DELAY
       MOV A,0
       MOV B,0
       LXI H,2004H
       CALL OUTPT
       LXI D,0FFFFH
       CALL DELAY
       JMP DPY
```

アドレス　データ
2000　10 0E 11 12 15 15 15 15

アドレス　プログラムデータ

```
2010  31 C8 20
2013  3E 01
2015  06 00
2017  21 04 20
201A  CD B7 02
201D  3E 00
201F  06 00
2021  21 00 20
2024  CD B7 02
2027  11 FF FF
202A  CD F1 05
202D  3E 00
202F  06 00
2031  21 04 20
2034  CD B7 02
2037  11 FF FF
203A  CD F1 05
203D  C3 1D 20
```

マシン語

この部分がパソコンのMONモードで使ったマシン語にあたるところです。このプログラムは2010番地からスタートします。

この例題のプログラムは、アドレス2010からスタートするようになっています。

アドレス(番地)データ

2010　31 C8 20
2013　3E 01

RESET キーを押すと、−8085を表示します。次に SUBST MEM キーを押すと、小数点がひとつ点灯され、あとは消えます。

−　80 85
↓
　　　.

ではプログラムしてみましょう。電源をON(オン)にすると、6桁全部フルコードの日となります。どうしてこうなるかというと、このモニタプログラムがこうなっているからです。

日.日.日.日. 日.日.

次に、新たにプログラムしたいデータ 3 1 NEXT と押すと、新たなデータにおきかえアドレスは次へ2011となります。

2010 31
2011 46
残っているデータ

方法は、キーを 2 0 1 0 と押し、NEXT と押すことで、2010番地とそのデータが表示されます。

2010
2010 C5
残っているデータ

ちょっとこのおかしな書き方について説明しておきましょう。
これは、2010−31
　　　　2011−C8
　　　　2012−20
と書くところを略して
2010 31 C8 20
としているのです。
どうしてこんな書き方をするのかということはもう少しあとで説明します。
数字(?)2桁でひとつの記憶単位になってます。

これだけのプログラムを入れるのに、何回キーを押したと思いますか? 149回です。ああ、しんど。

あとは、同じことのくり返しです。順々に書き込んで 2 0 NEXT でプログラムの書き込み完了です。

203D　C3 1D 20

203F 20

2進
0011 0001
16進
31
同じなのよ。

このあとで書き込んだ8桁分のデータは文字として表示させるためのものです。

アドレス データ

アドレス	データ	
2000	10	… H
2001	0E	… E
2002	11	… L
2003	12	… P
2004	15	… ブランク
2005	15	… 〃
2006	15	… 〃
2007	15	… 〃

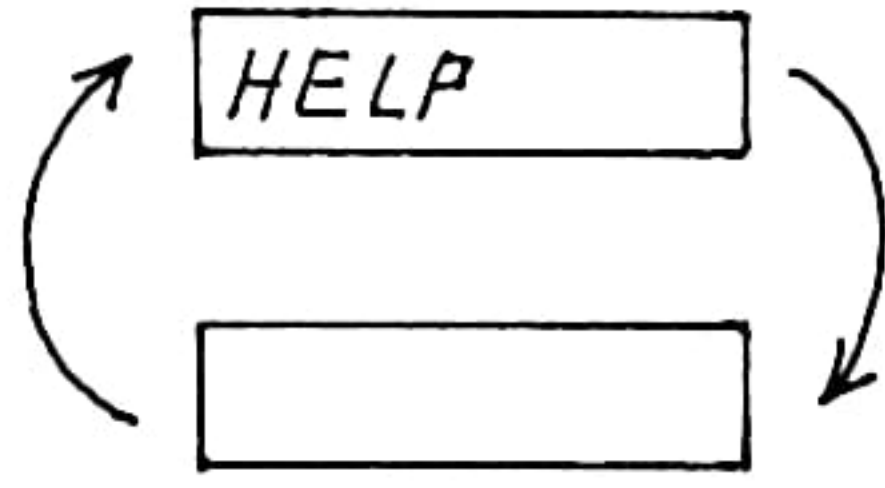
交互に表示させるとHELPが点滅することになる

RESET SUBST MEM 2 0 0 0 NEXT 1 0 NEXT と順に押して、アドレス2007までのデータを書き込みます。

2000 10

でも、この例題では、さらにプログラムで使うデータも書き込めと書いてあります。

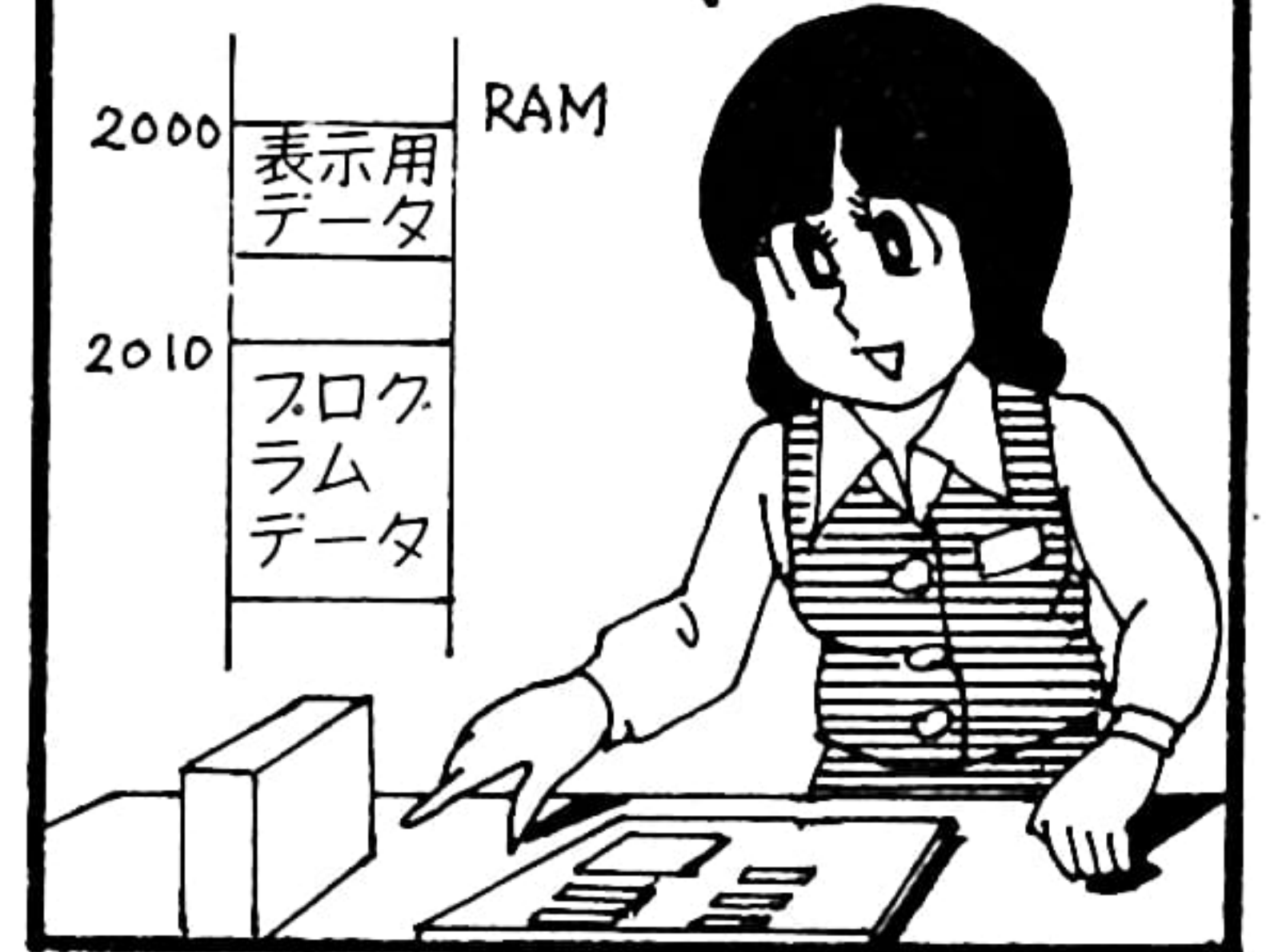
2000　10 0E 11 12 15 15 15 15

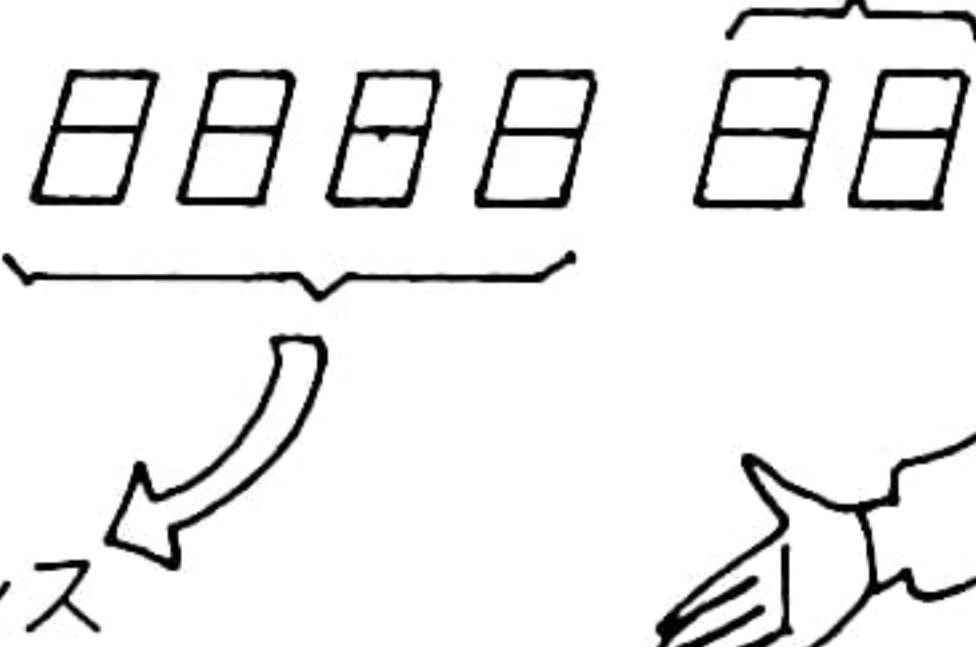
このプログラムは、グルグルと同じ所を回っているので、STOPさせたい時は、RESETキーを押します。

HELP

RESET GO 2 0 1 0 EXEC と順に押すことにより、プログラムが実行され、HELP、HELPと点滅します。

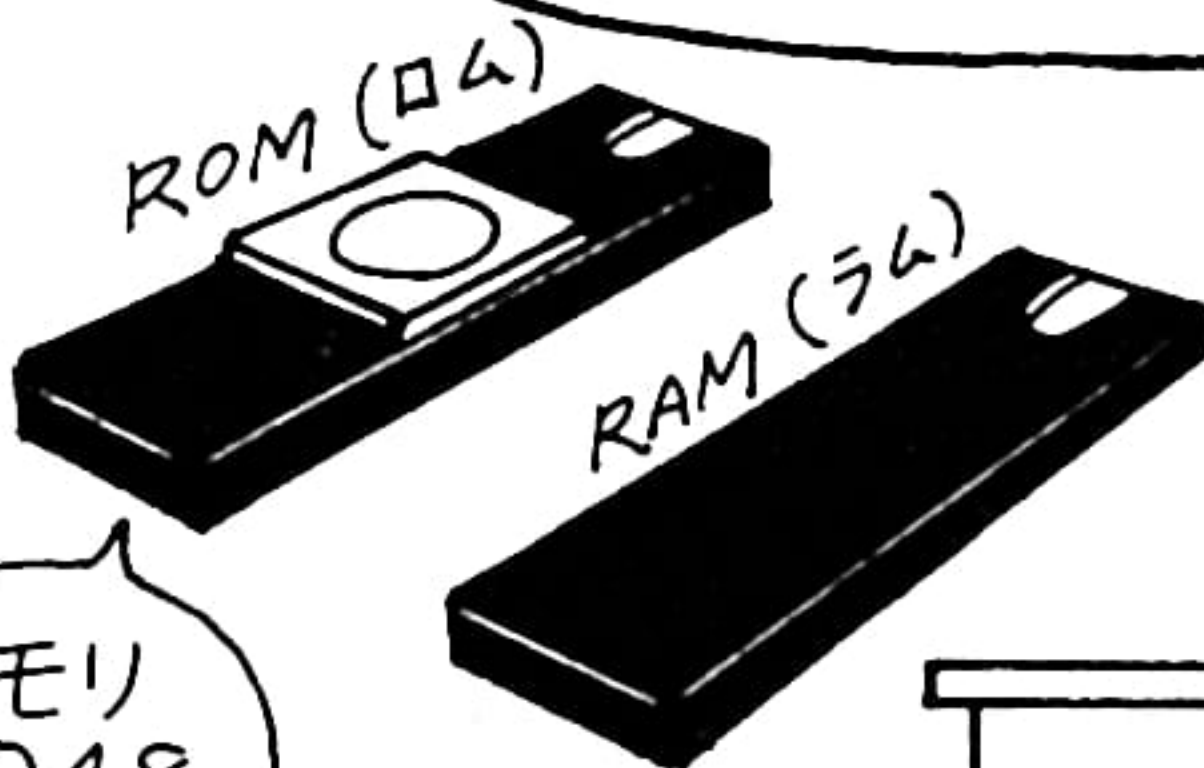
HELP

これでプログラム書き込みが完了しました。いつでもプログラムの実行が可能です。

2000
表示用データ
2010
プログラムデータ
RAM

1バイト
データ
メモリ
アドレス

ここで、アドレスとメモリについてちょっと触れておきましょう。

これで、マニュアルの例題'HELP'プログラムの説明は終わりです。

HELP
HELP

このキットには、2種類のメモリがついています。ひとつはROM、もうひとつはRAMです。

ROM(ロム)
RAM(ラム)
メモリ2048バイト
メモリ256バイト

メモリには必ずアドレスがついており、8bit CPUの場合、16bit のアドレスを持っています。メモリがついていない時は(空間)となります。これをアドレス空間といい、0000からFFFFまでです。

下位アドレス
0000
FFFF
65536メモリ分

このうち、私たちがプログラムで使えるメモリは、RAMが使用されている2000～20FFの256アドレス分だけなのです。

この2つのメモリは、ハード的にそれぞれのアドレスが割り当てられています。

F F F F
1111 1111 1111 1111
16ビットすべてON

メモリの入っていない部分

下位アドレス
2048バイト ROMメモリ
空間
256バイト RAMメモリ
空間
0000
07FF
2000
20FF
FFFF
ユーザーが使えるメモリエリア

電源をONにすると、CPUはアドレス0000からスタートします。ここに、モニタプログラムが入っているのです。

電源ON

8.8.8.8. 8.8. 表示

ROM
モニタプログラム
0000
07FF

私たちが使えないメモリなんて意味がないと思うのですが、実はこれにはすでにプログラムが入っているのです。

プログラム済

ROM メモリ

もし、このモニタがなかったら、つまり最初から自分で全部やろうと思ったら、

私たちが、16進数を使ってプログラムできるのも、このモニタプログラムのおかげです。

16進キー
コマンドキー
受付けます

モニタ
ROM

電源をONした時に、LEDが全桁点灯したのは、このモニタプログラムが実行されたからなのです。

8.8.8.8. 8.8.

この時のデータ表現は2進数、とてもやってられません!!

0010 0001
0100 1100

お手上げ

アドレス16ビット分のスイッチ16個を、アドレスバスにつなぎ、データ8ビット分のスイッチ8個をデータバスにつないで、ひとつひとつプログラムデータをメモリに書き込まなくてはなりません。

アドレス16ビット
データ8ビット
WRITE SWITCH

0010 0000 0001 0000
2 0 1 0

0011 0001
3 1

アドレスをセットして…
データをセットして…
書き込みスイッチON

パソコンのモニタモードは、こうしたワンボードマイコンユーザーにとってはまさに福音だったのです。パソコンにはCRTがついているし、はるかに使いやすくなったのです。

モニタプログラム

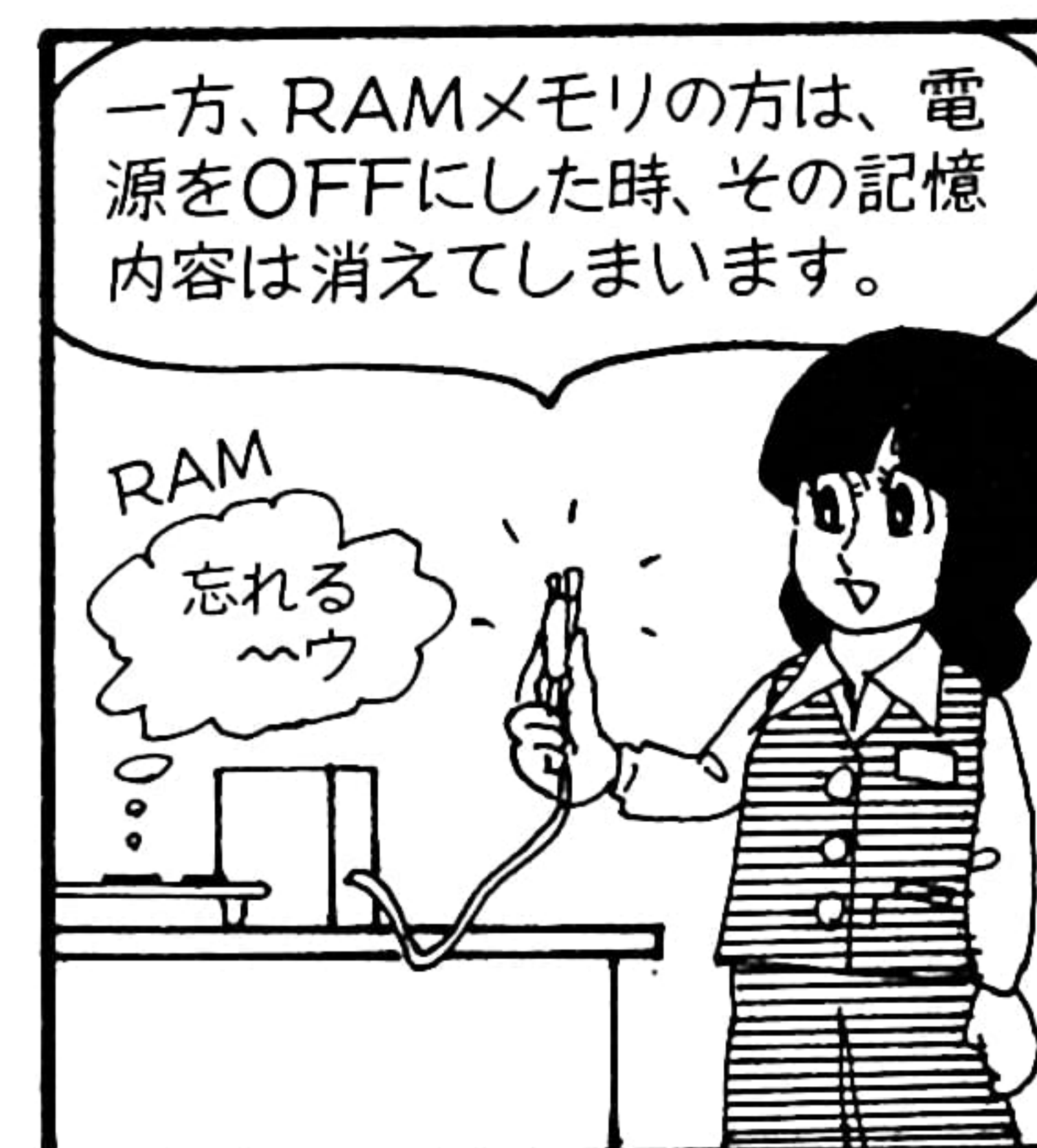
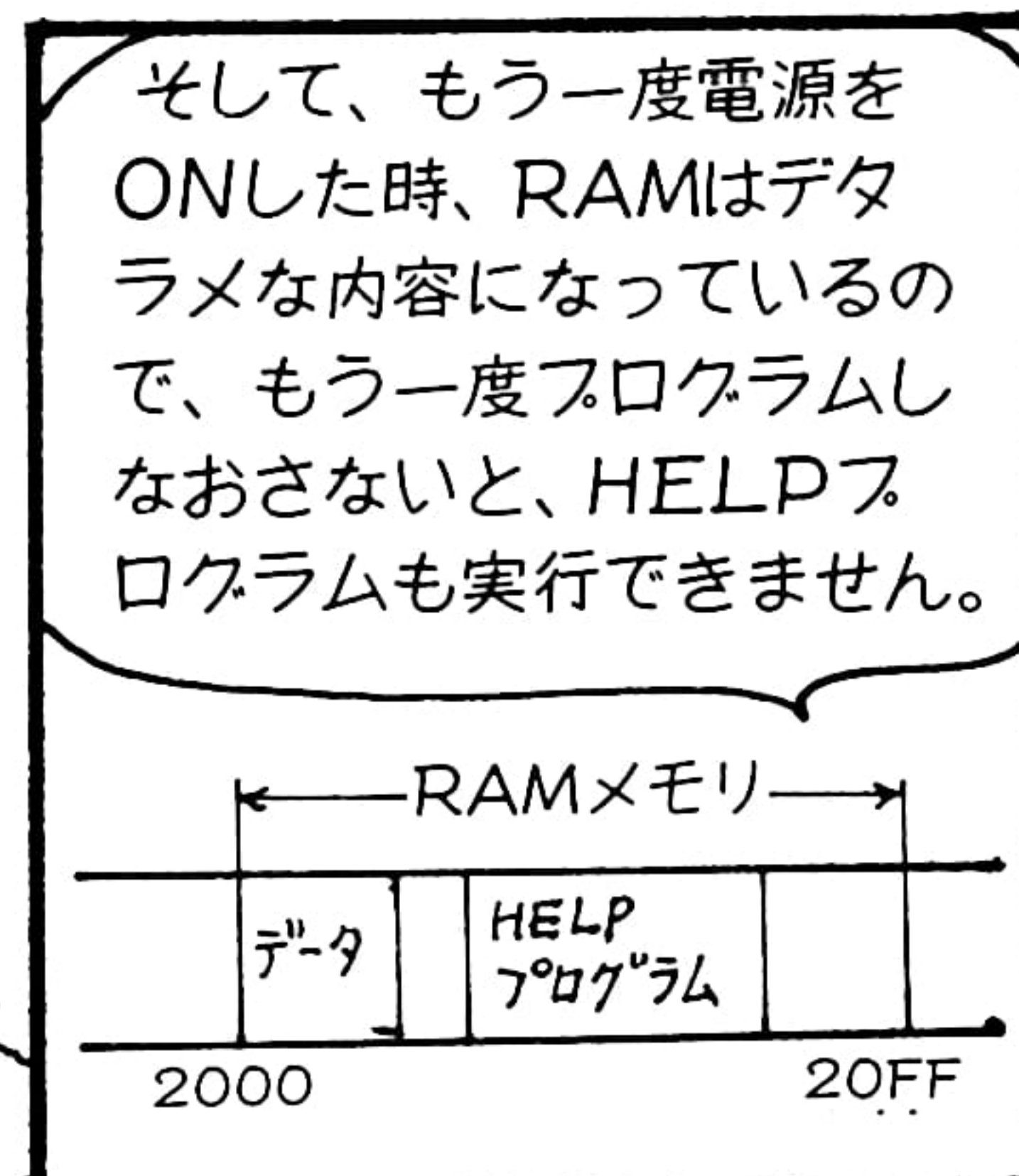
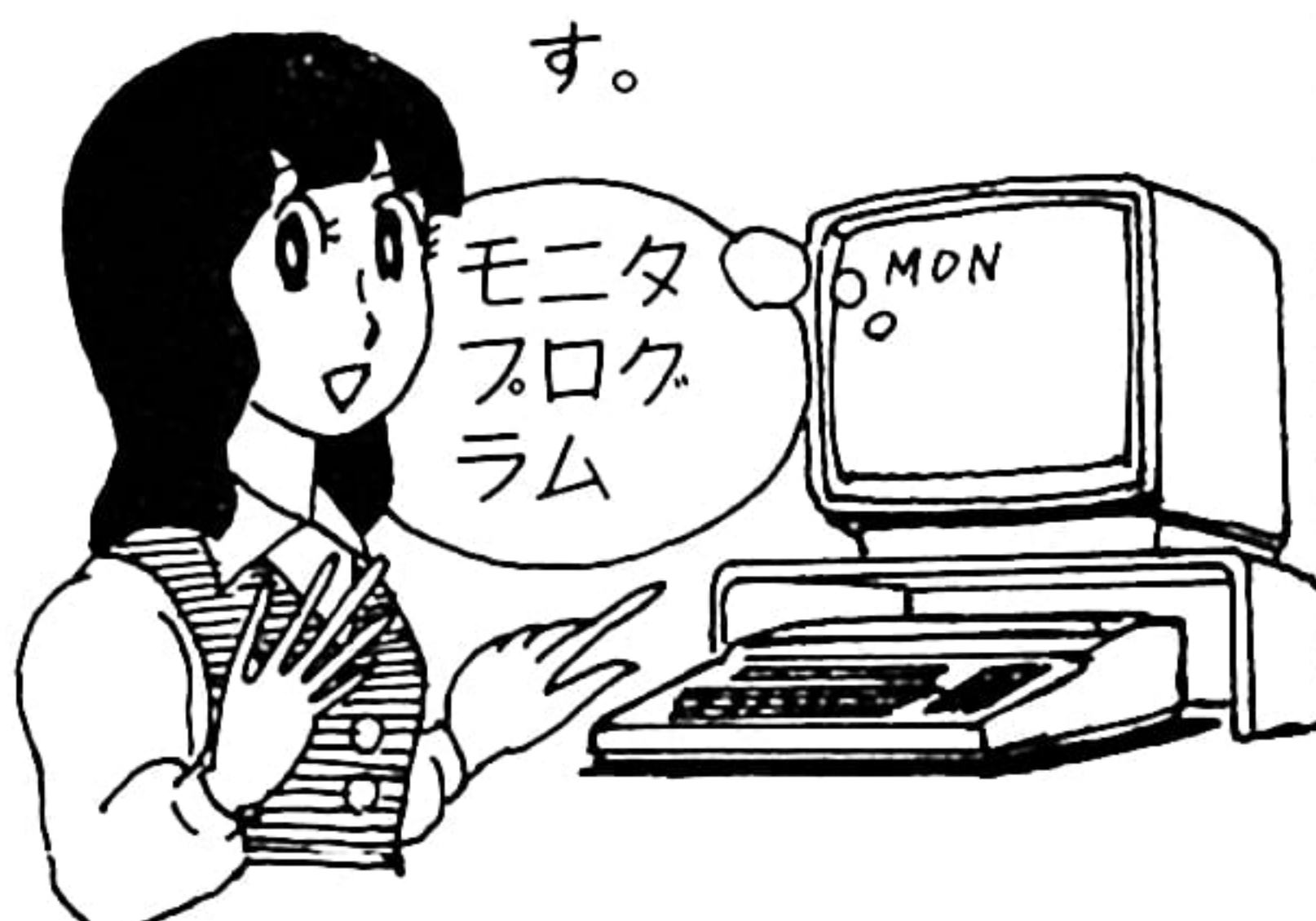
MON

そして、もう一度電源をONした時、RAMはデタラメな内容になっているので、もう一度プログラムしなおさないと、HELPプログラムも実行できません。

RAMメモリ
データ
HELPプログラム
2000
20FF

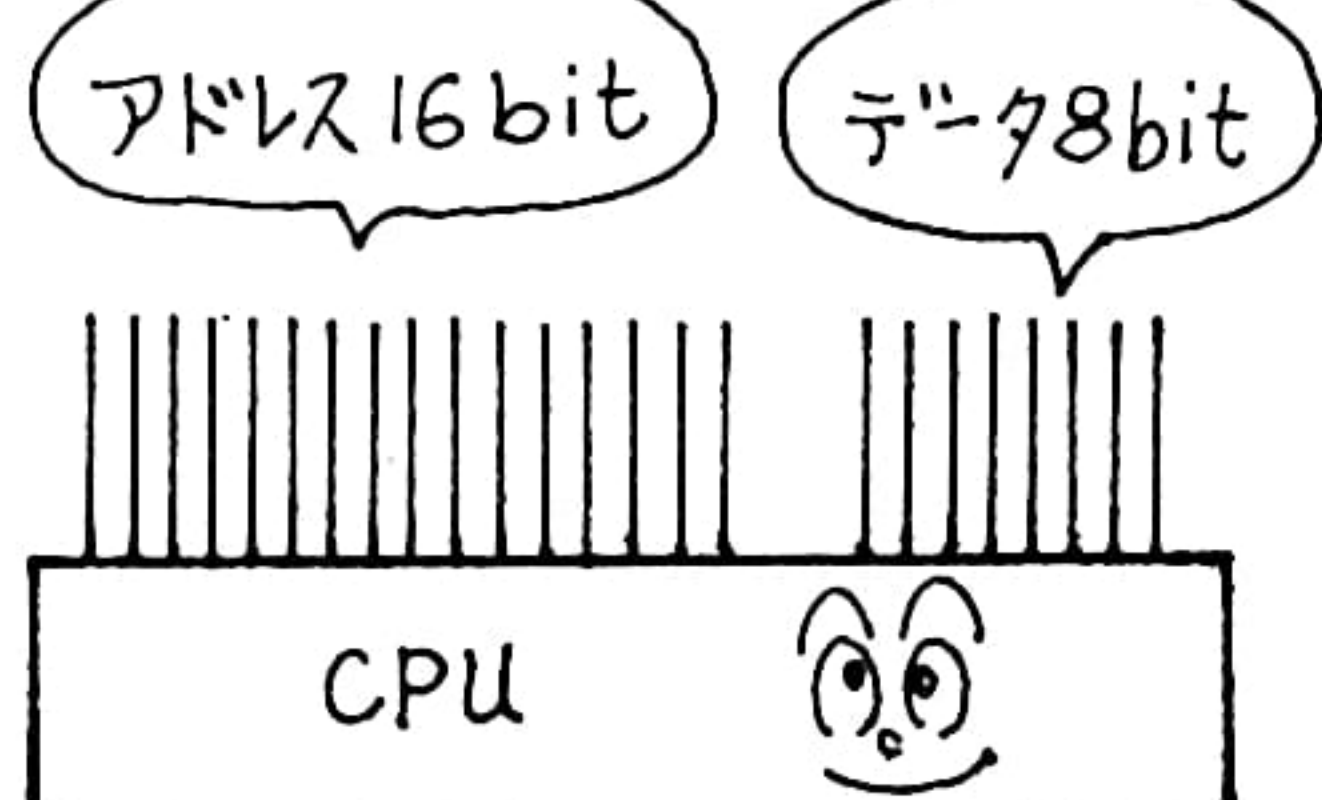
一方、RAMメモリの方は、電源をOFFにした時、その記憶内容は消えてしまいます。

RAM
忘れる〜ウ

③マシン語の『いろは』は0(ロー)か1(ハイ)

コンピュータが使うのは、2進数です。このことは大型、小型に関係ありません。

アドレス16bit

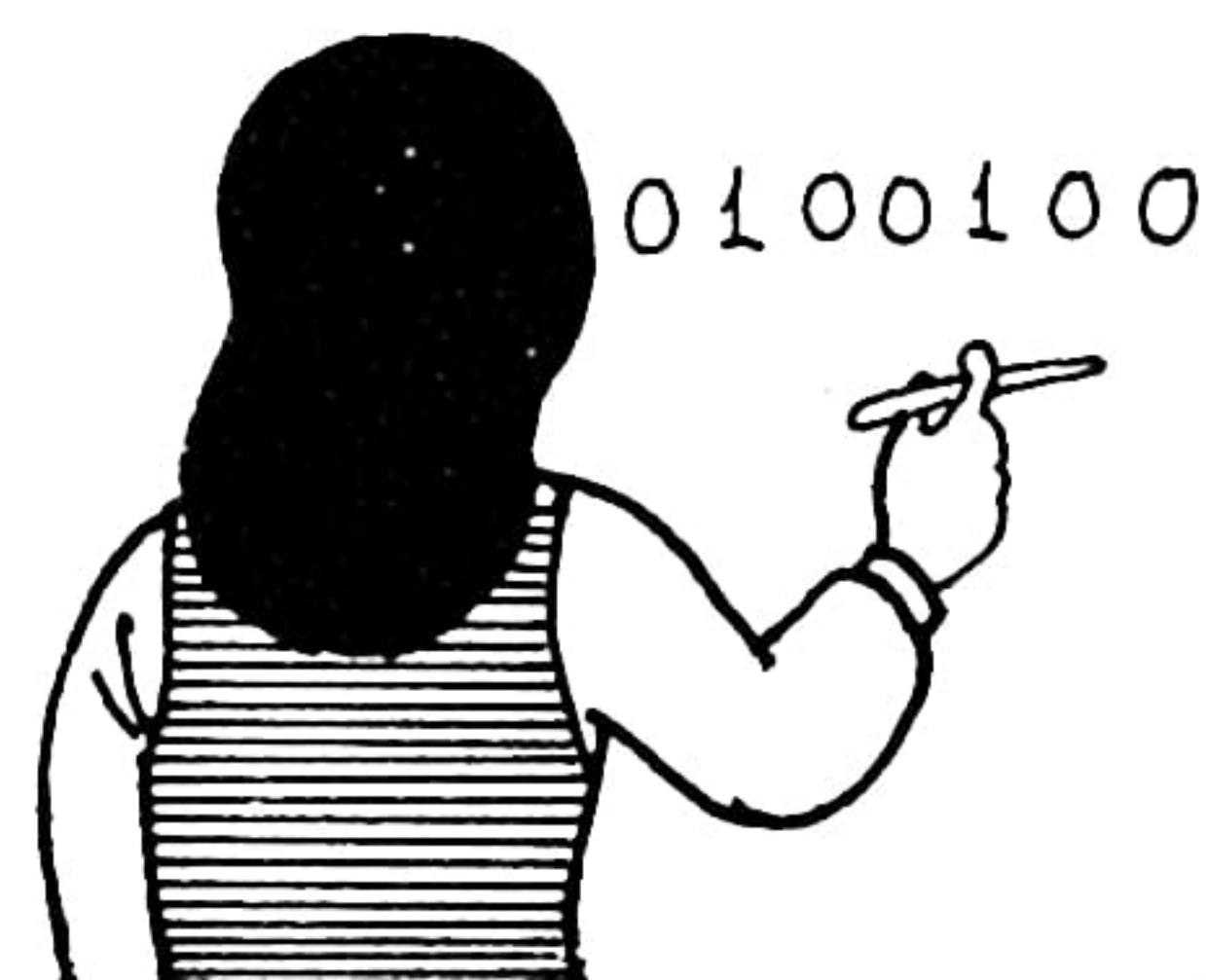
データ8bit

CPU

私たちが日常使っているのは10進数ですね。2進数も16進数もなじみのない数字です。

1000円

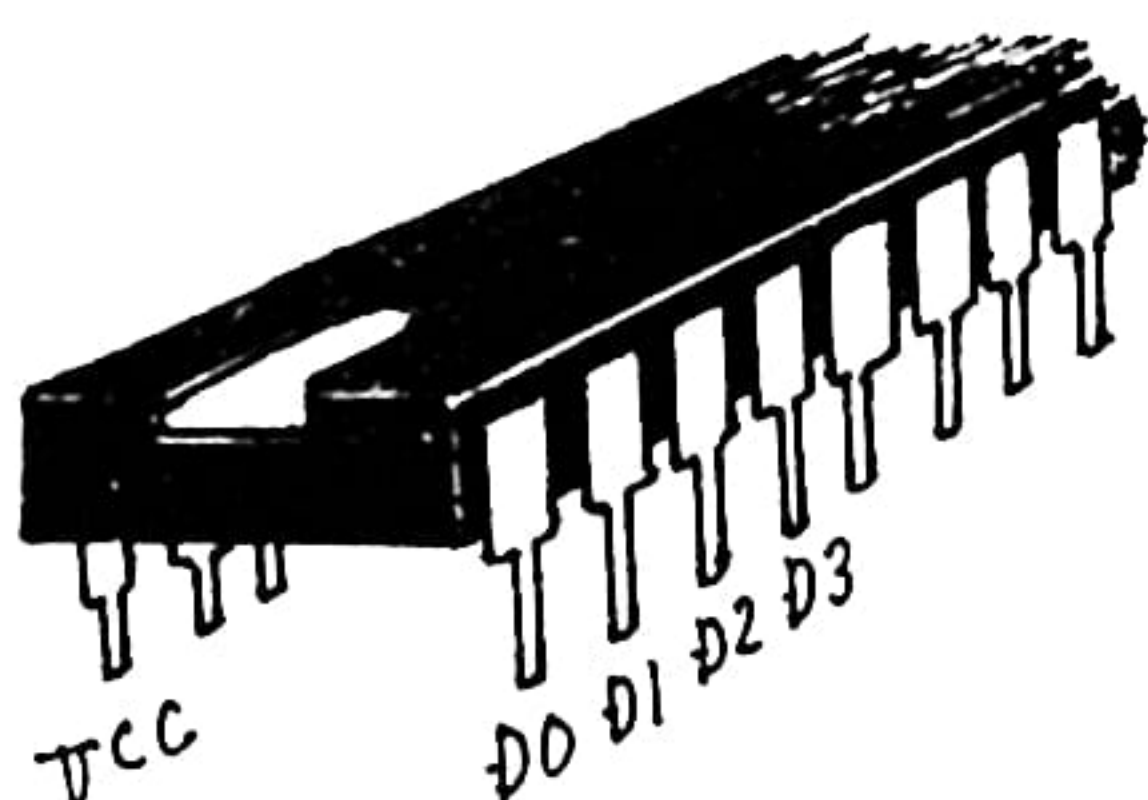
10m

100点

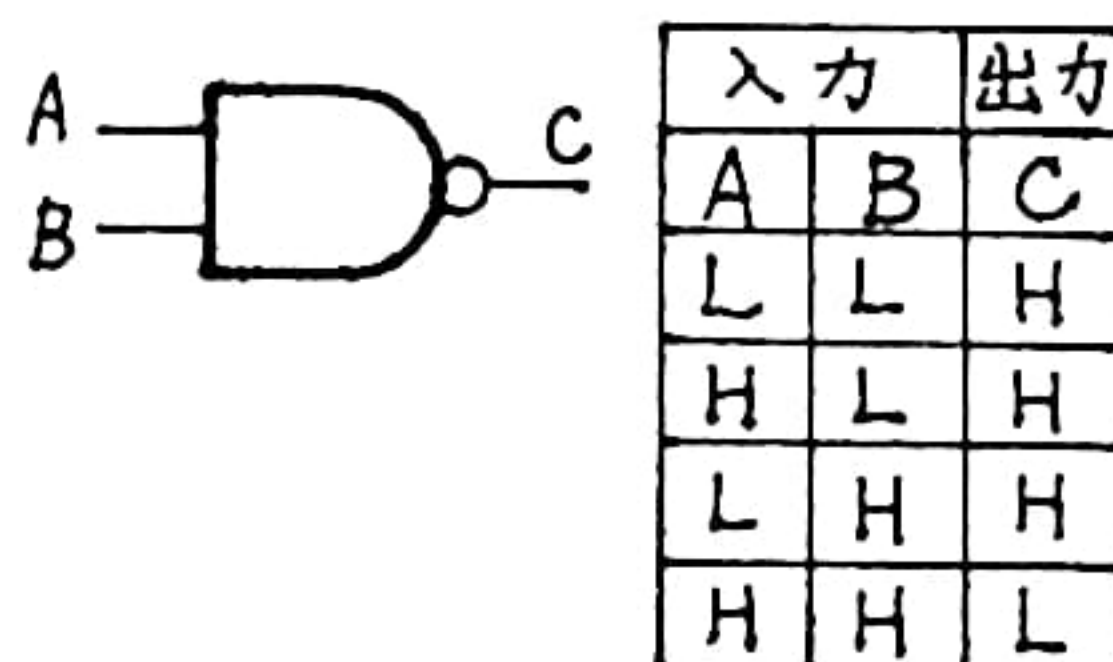
100kg

0100100

ICの信号ピンには番号がついていて、各ビットの桁がわかるようになっています。

VCC

D0 D1 D2 D3

ふつう、bit は0か1で表現しますが、ICなどのロジック回路では電圧信号として、HとかLとかで表現することもあります。

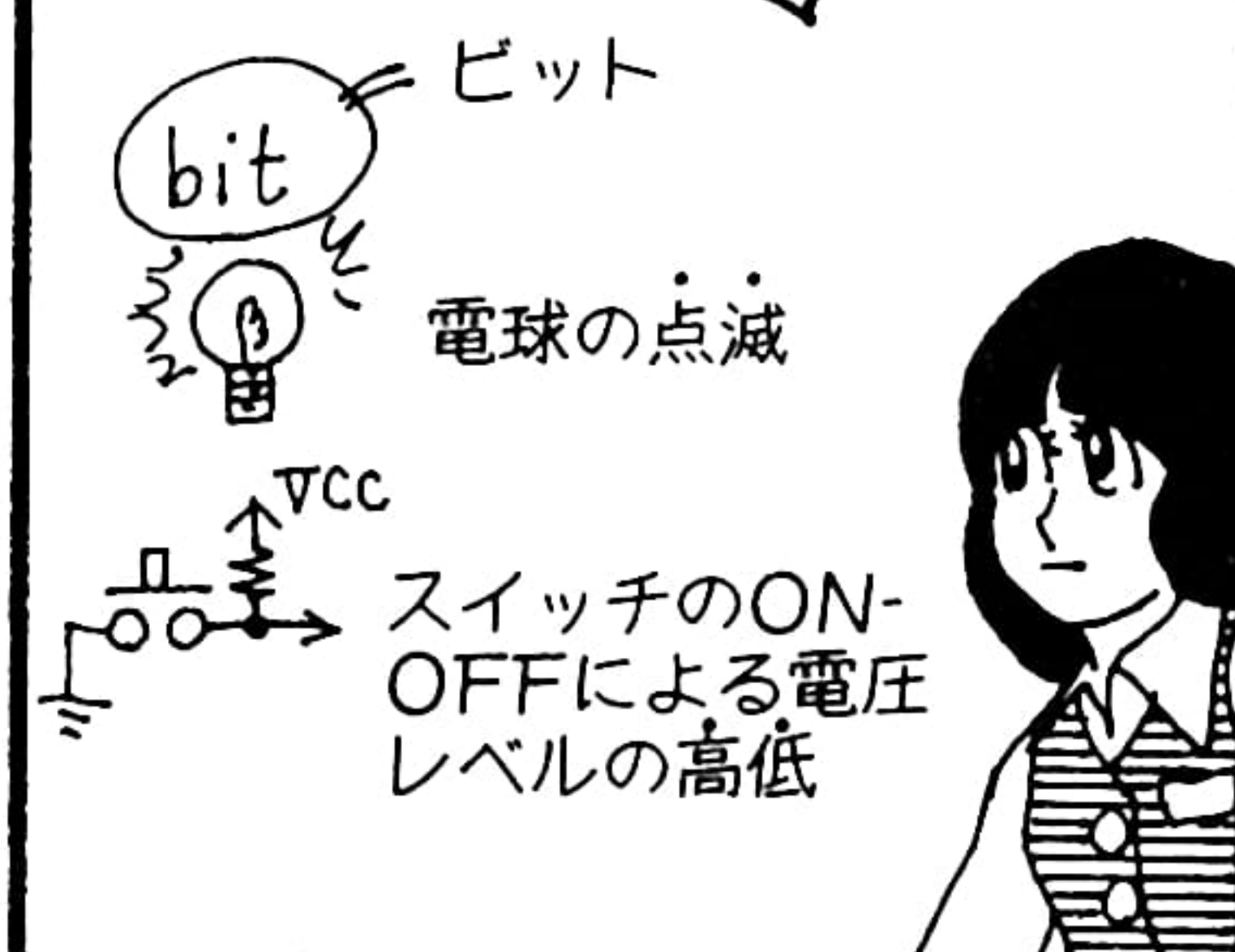
A B C

入力		出力
A	B	C
L	L	H
H	L	H
L	H	H
H	H	L

2進数の最小単位が1ビット(bit)と考えていいと思います。

bit ビット

電球の点滅

VCC

スイッチのON-OFFによる電圧レベルの高低

4ビットでは、1と0で16通りの区別ができます。

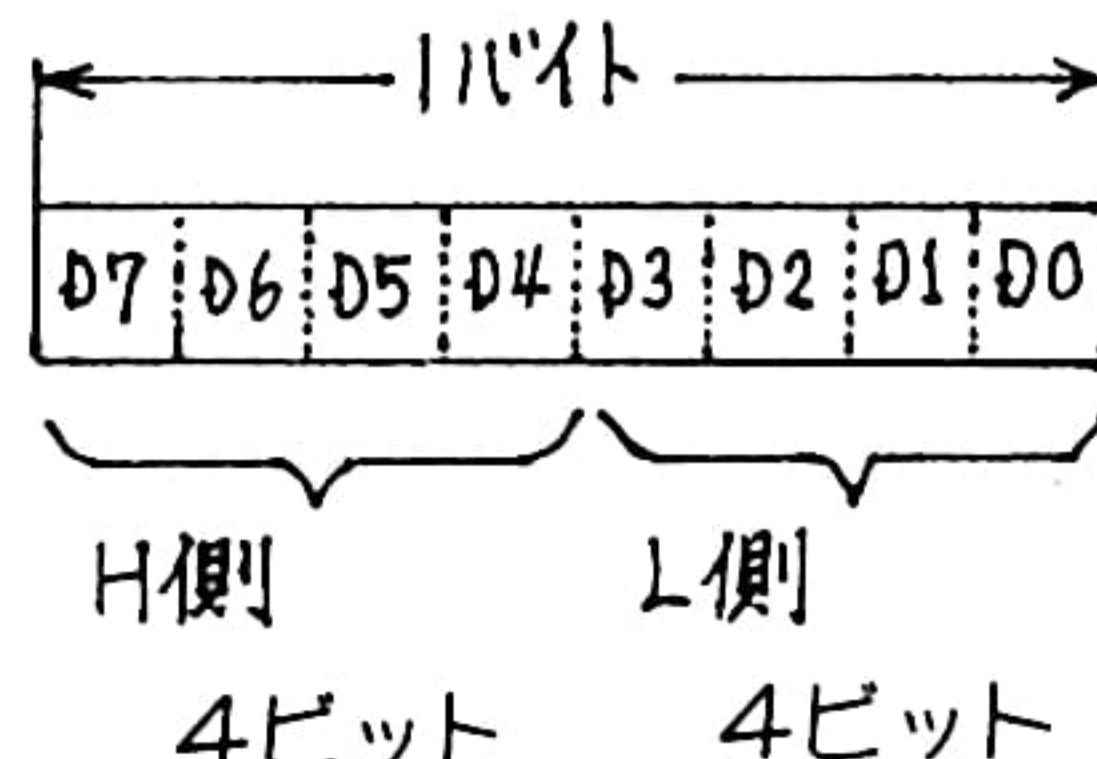
	D3	D2	D1	D0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

マイコンで使われる機械語は、1バイトの半分の4ビットで区切ったものを使っています。

1バイト

D7 D6 D5 D4 D3 D2 D1 D0

H側 4ビット

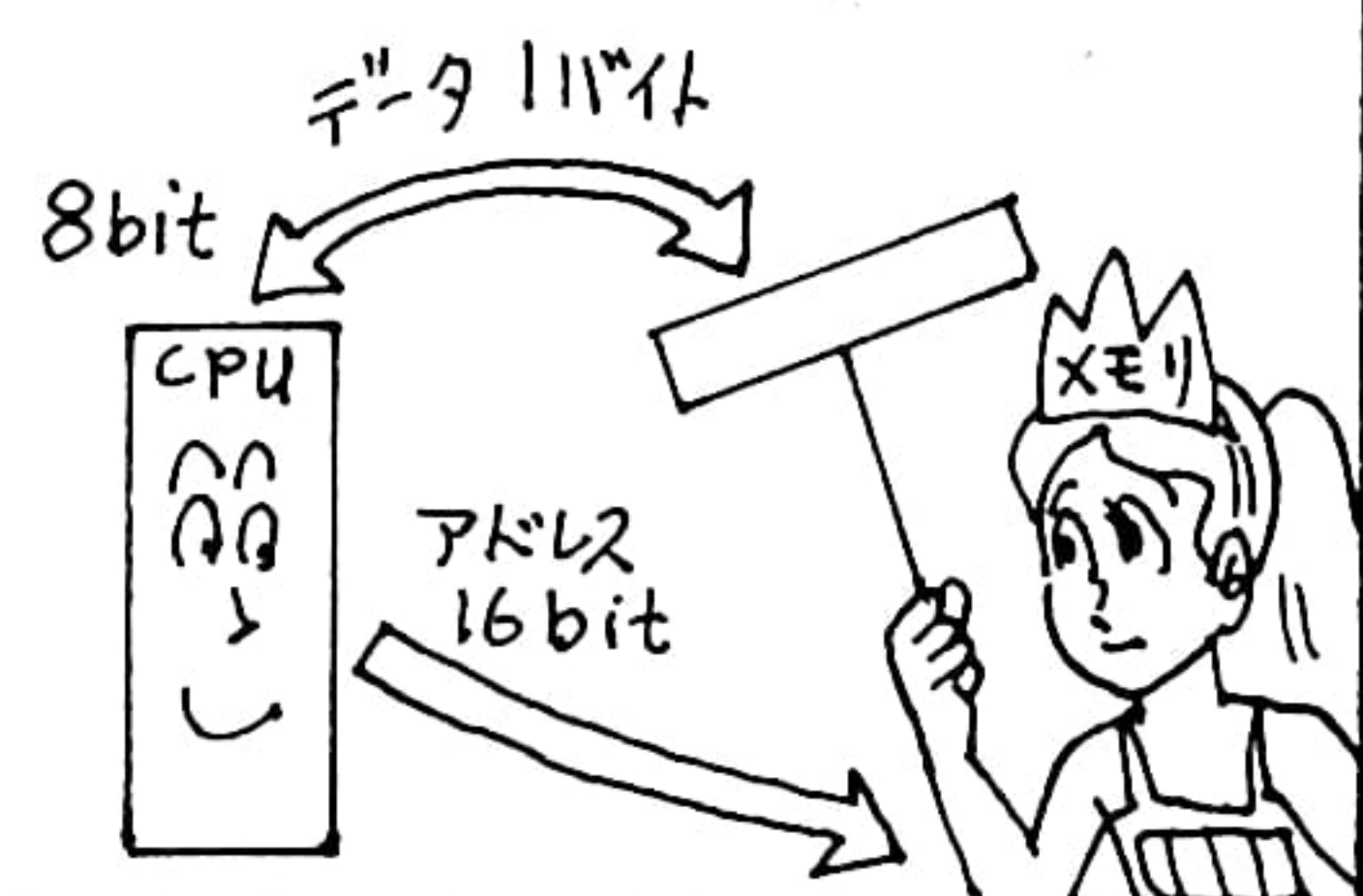
L側 4ビット

8bit CPUの場合、メモリも1アドレス8ビット単位が便利です。8ビット単位の情報を1バイト(byte)といいます。

データ 1バイト

8bit

CPU

メモリ

アドレス16bit

CPUは2進表現以外受け付けませんので、人間の方で16進数を使えるプログラムをつくるしかありません。

00110001

モニタプログラム

31

8ビットによる2進表現よりも、16進2桁表現の方が私たちには見やすく、プログラムもしやすいのですが……。

2進8ビット表現	16進2桁表現
0011 0001	31
1100 1000	C8
0010 0000	20

00110001

11001000

00100000

31

C8

20

2進数の場合は、0か1のみの組合せによる表現となり、
1 0

おなじみの10進数については、0～9までの10個の記号の組合せ。
0 1 2 3 4 5 6 7 8 9

そして16進数については、0～9まではそのままで、A、B、C、D、E、Fの記号を追加しました。
0 1 2 3 4 5 6 7 8 9 A B C D E F

これら3つにはそれぞれ独立した記号を使えばよかったのにと思いますが…。
意味が違うのに同じ記号を数字として使うからまぎらわしいのよ。

さて、マイコンの世界のように、2進、10進、16進が混在していると、たとえば1010という数字は何に属したものかわかりません。
2 バイナリー
10 デシマル
16 ヘキサデシマル
1010
?

そこで、数字の後にB、D、Hをつけて区別するようにしています。
2進数……1010B
10進数……1010D
16進数……1010H

とくに16進の場合で、最初の1文字がA、B、C、D、E、Fのいずれかの場合は、その頭に0(ゼロ)をつけて英文字と区別するようにしています。
ゼロ
0A5H
とくにアセンブラ言語として使う時

ためしに全部10進数に直してみました。
あとででてくるがアセンブラを使う時にこの考えが必要。
$1010B=(2^1+2^3)D=10D$
1010D　これは10進だからそのまま
$1010H=(1\times16^1+1\times16^3)D=4112D$
どうです？　それぞれ10進数の10、1010、4112となりました。

このうちマイコンの世界で最もひんぱんに使われるのが2進と16進の相互変換です。
バイナリー2進
ヘキサデシマル16進

練習すれば誰でもそれなりに上達します。
E3っ
4つずつ区切るのがコツ
1101 0111

8ビットマイコンでは、アドレス16ビット、データ8ビットで情報交換します。

0000000010100111
アドレス
データ
10101001

ですから、私たちも、2進表現と16進表現が区別なく理解できる必要があります。

00A7
アドレス
データ
A9
文書にするときは0A9Hと表現する

私たちマイコンを使うものがマシン語として使うのは、主に16進数です（2進表現と同じとみなします）。

堅苦しいことを言うと、CPUは2進数しか扱えないのですが、それでは私たちもデータの表現に不便です。

表現の自由 賛成

16進数表現のほうがスマートでわかりやすいのです。

1010 1001
同じ
A9
メモリ
0000000010100111
同じ
00A7

マイコンの高度な活用を考えている人は、ぜひ16進数に慣れましょう。

次にアドレスとデータについて見てみましょう。

私、CPUのことはまたあとでお話します。

OUT
モニタ
データ
モニタプログラム
IN
アドレス

アドレスは、0000~FFFFの範囲でハードウェア設計時に決まってしまうもので、その後に変更することはできません。

メモリ
アドレス指定は一方通行

これに対して、データはいつでも任意に変更できるわけです。ハード的に同じでも違う働きをすることができます。つまり、ソフトウェアに属するものです。

メモリ
ただしROMは出力のみ

そうそう。メモリにはROMとRAMがありましたね。ここでもう少しお話しておきましょう。少しずつマシン語に近づいてきました。

CPU
ROM
RAM
メモリ

このデータとアドレスとはこれから長いつきあいになります。

プログラム
メモリ
メモリ
メモリ
CPU
実行

実装されたあとは、読出し専用の一方通行で使用します。書き込み命令には使えません。

ROM

ROMちゃんには、あらかじめデータを書き込んでおきます。

01011101
5
ROM
メーカー

ここで、ROMちゃん、RAMちゃんをご紹介します。

1バイト
ROM
RAM

たとえばパソコンでMONモードにしておいて0000～5FFF間でデータを書き込んでみてください。

```
OK
MON
*S 4000
4000 4C-
```

一方、RAMちゃんは、ユーザープログラムのデータエリアとして、また定数データエリアとして読み、書きができます。

双方向データ
RAM
ユーザー
2010 31C820 3E01

これはPC-8001の場合です。FM-7やその他のパソコンではROM、RAMエリアは違っているはずです。マニュアルのメモリマップを見てください。

これに対して C000～FFFF番地はRAMなのでデータを書き込めばメモリの内容が変わります。

```
*D C000
C000 EF 32 3E 4E
*S C000
C000 EF-01 32-02
*D C000
C000 01 02 3E 4E
```

変わっている

Dコマンドでメモリ内容を見るとデータはもとのままです。CPUは書き込み手続きをしたのですがメモリが受け付けなかったのです。ROMメモリだからです。

```
OK
MON
*S 4000
4000 4C-01 C1-02
*D 4000
4000 4C 01 C3 75 3F
```

変わっていない

ではこのモニタプログラムを使って私たちはどこまでマシン語を知ることができるのでしょうか。

突然オペレーションコード（略してオペコード、またはOPコード）という言葉が出てきますが、またあとで説明しますので軽く読みとばしてくださいね。

これは、HELPプログラムの最初の部分です。

番地	データ
2010	31C820
2013	3E01
2015	0600
2017	210420
201A	CDB702
⋮	⋮

各アドレスのメモリには、このように記憶されます。

	×	×
2010	3	1
2011	C	8
2012	2	0
2013	3	E
2014	0	1
2015	0	6
2016	0	0
2017	2	1
2018	0	4
2019	2	0
201A	C	D
201B	B	7
201C	0	2

私たちはこのキットでは256人います。

RAM

201C

このプログラムでは、2010番地の31というデータがスタートコードになります。

31

RAM

プログラムスタートの時、2010を指定すると、CPUはこの番地のデータをオペレーションコードとして読み込みます。

スタート

モニタプログラム

アドレス

2010

2010番地から読み出したデータは31です。このオペレーションコードの意味は、

オペコード

31

2010

スタックポインタを、あとに続く2バイトで示される番地に設定するという命令です。

スタックポインタ 20C8

C8

20

2012

2011

この31というオペレーションコードは、あとに2バイトのデータが必要で、3バイトのセットで使われますので3バイト命令といいます。

3回読む

1st

2nd

3rd

31

このあとオペコードとして読むのは、次のアドレス2013のデータということになります。

もうモニタはいらない。

アドレス

2013

このデータは3Eでオペコードとしての意味は、

オペコード

3E

3 E

2013

次に続く1バイトのデータをCPUの中のAレジスタに読み込めという2バイト命令です。あとはこの調子で実行します。

Aレジスタ

01

0 1

2014

モニタプログラムでは、アドレスを1バイト書き込むごとに+1していました。

+1(インクリメント)

でも同じプログラムデータの中で全部が全部CPUへの命令でないことはなんとなくおわかりでしょう。

番地	データ		
2010	31	C8	20
2013	3E	01	
2015	06	00	
2017	21	04	20

CPUへの命令コード

そうでないコード

今の説明ではスタックポインタなどの言葉がでてきてチンプンカンプンだったかもしれませんネ。

そのため、プログラムデータは、これを打ち込む人間の方で十分に注意する必要があるのです。

本当のオペコードかただのコードかの区別はできません。

OPコード

OPコード

OPコード

31 08 20 3E 01

2010

CPUは、最初のOPコード解読から次々に連続してデータを読み込んでいます。

31は3バイト命令

CPU

OP

OP

OP

メモリ

OPコード一覧表

このことはROM型パソコンのMONモードでも同じことが言えます。暴走すると、プログラムが崩れてしまう恐れがあるため事前にLコマンドを使ってテープにしまっておく方が無難ですよ。

HELPプログラムは例題としてメーカーの人が作っておいてくれたもので、正しくインプットすればすぐ実行できます。

もし途中、OPコードを間違って入れたり、データが不連続だったりすると、実行時正しく働きません。

……

……。

暴走中

自分でこのマシン語を使ってプログラムしてみたい時はいったいどうしたらいいのでしょうか?それにはCPUのしくみを知らなくてはなりません。

CPU

ところで31C8203E…とならんでいる数字がマシン語らしいということは、よくわかったのですが、

マイコン雑誌などに載っているマシン語リストを見ながらキーインする時は数字ひとつも間違えないように注意深く行なうことです。

FM-7/8用

○○インベーダー

4000 10 CE 6F FF 86

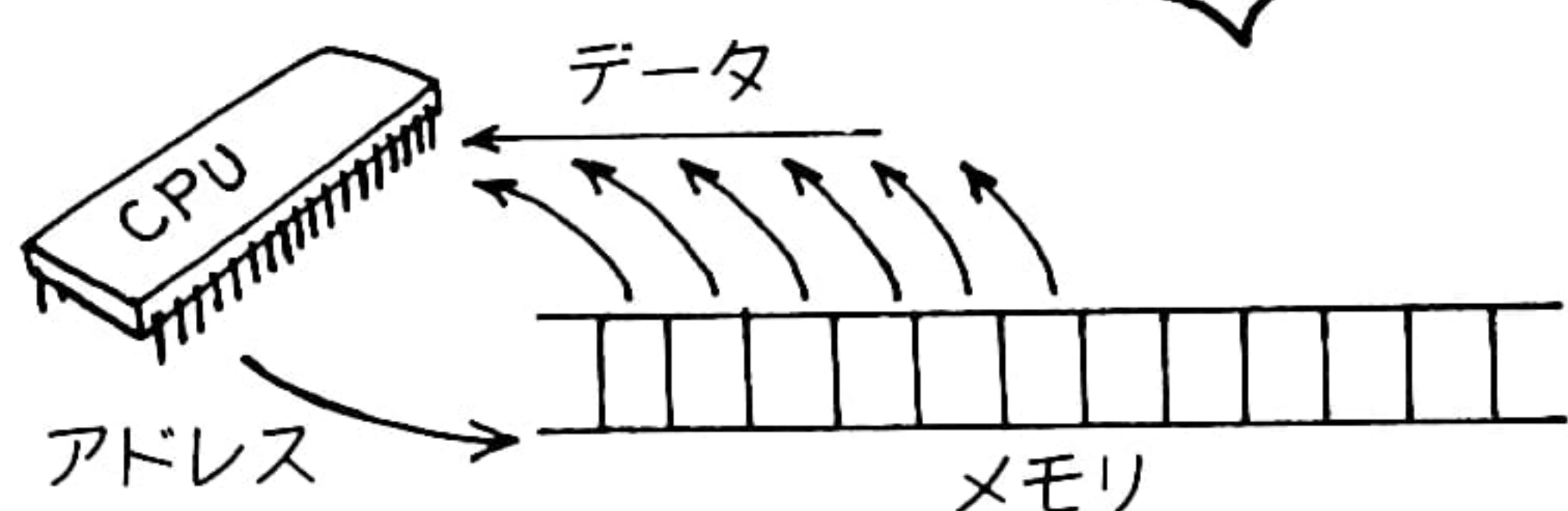
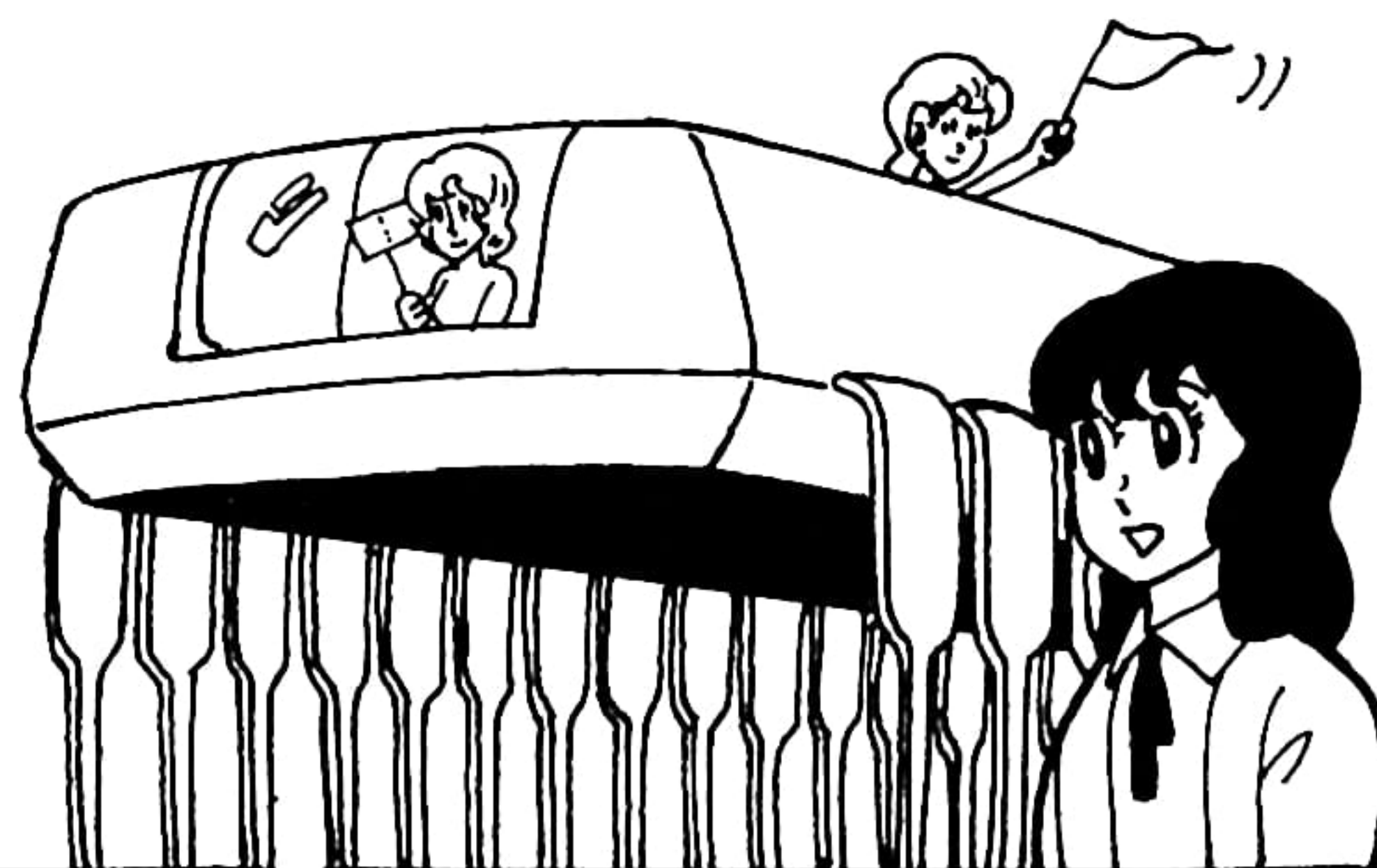
4010 4B 20 17 00 D

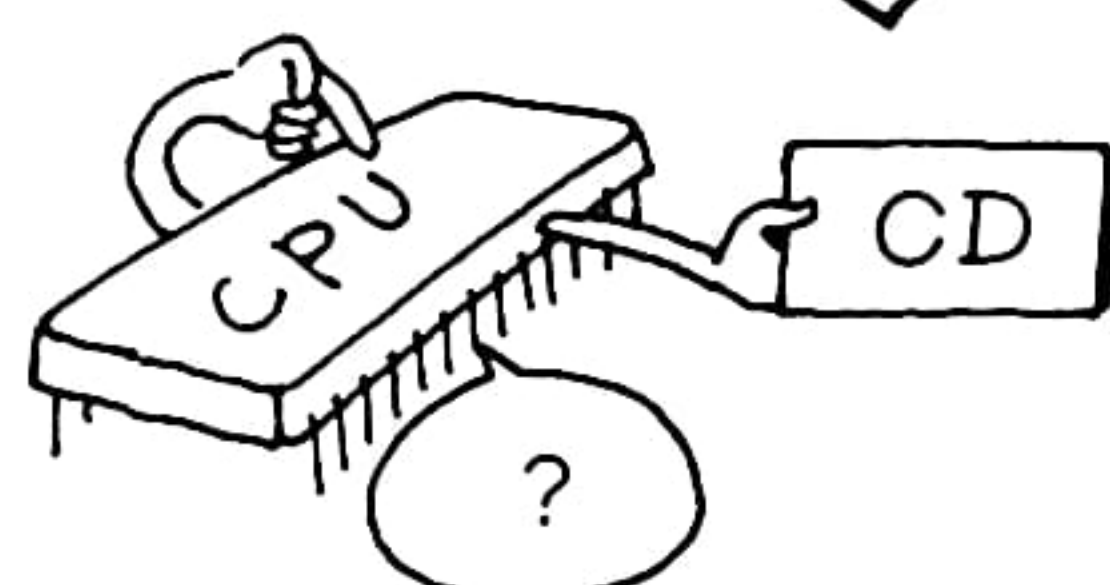
96 3D 2A 1E

DB 3C D7

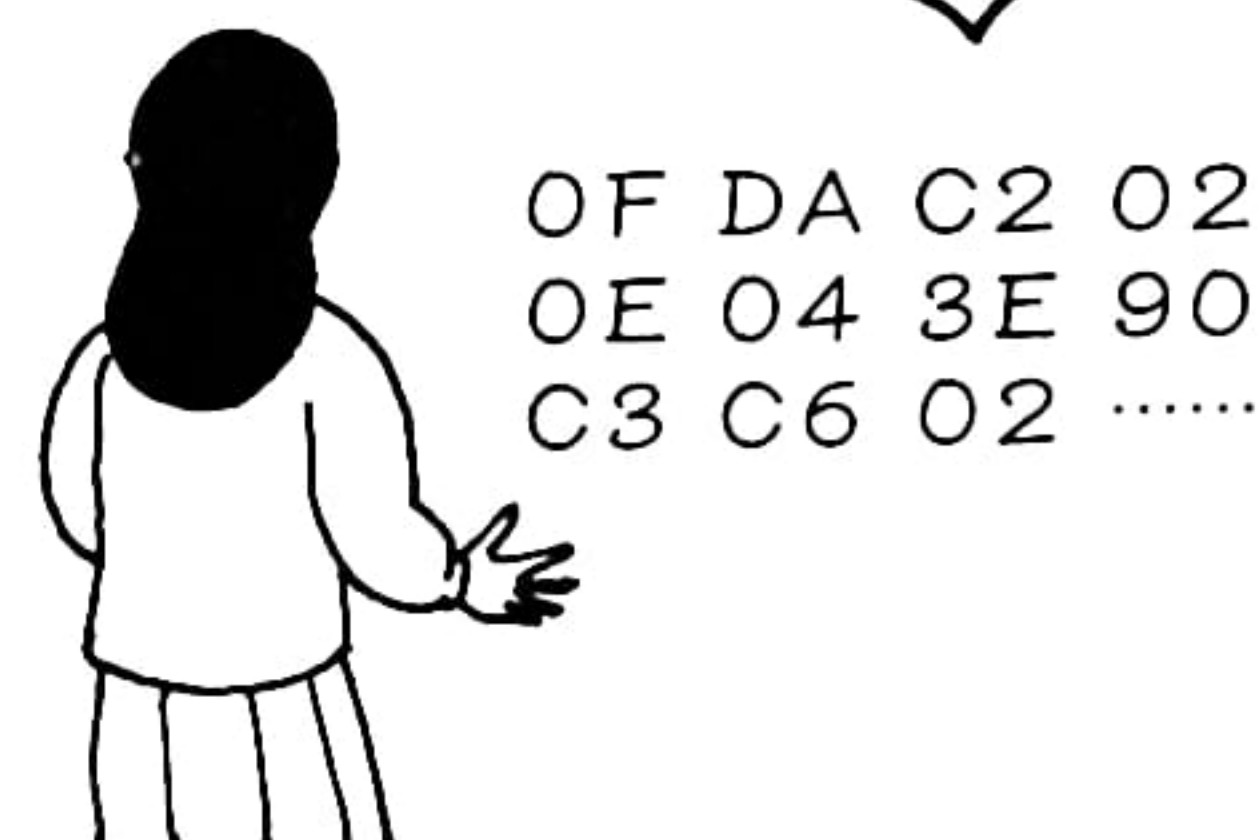
④CPUのしくみ (8080の場合)

CPUというのはメモリに入っているデータを読み込んでその内容を解読して所定の処理を行ない、また次のメモリに入っているデータを読む……といったことをくり返すようにできています。

データ
CPU
アドレス
メモリ

そのためには00、01、02……FD、FE、FFといったコードがCPUにとってどんな意味を持つものなのかを知らなくてはなりません。

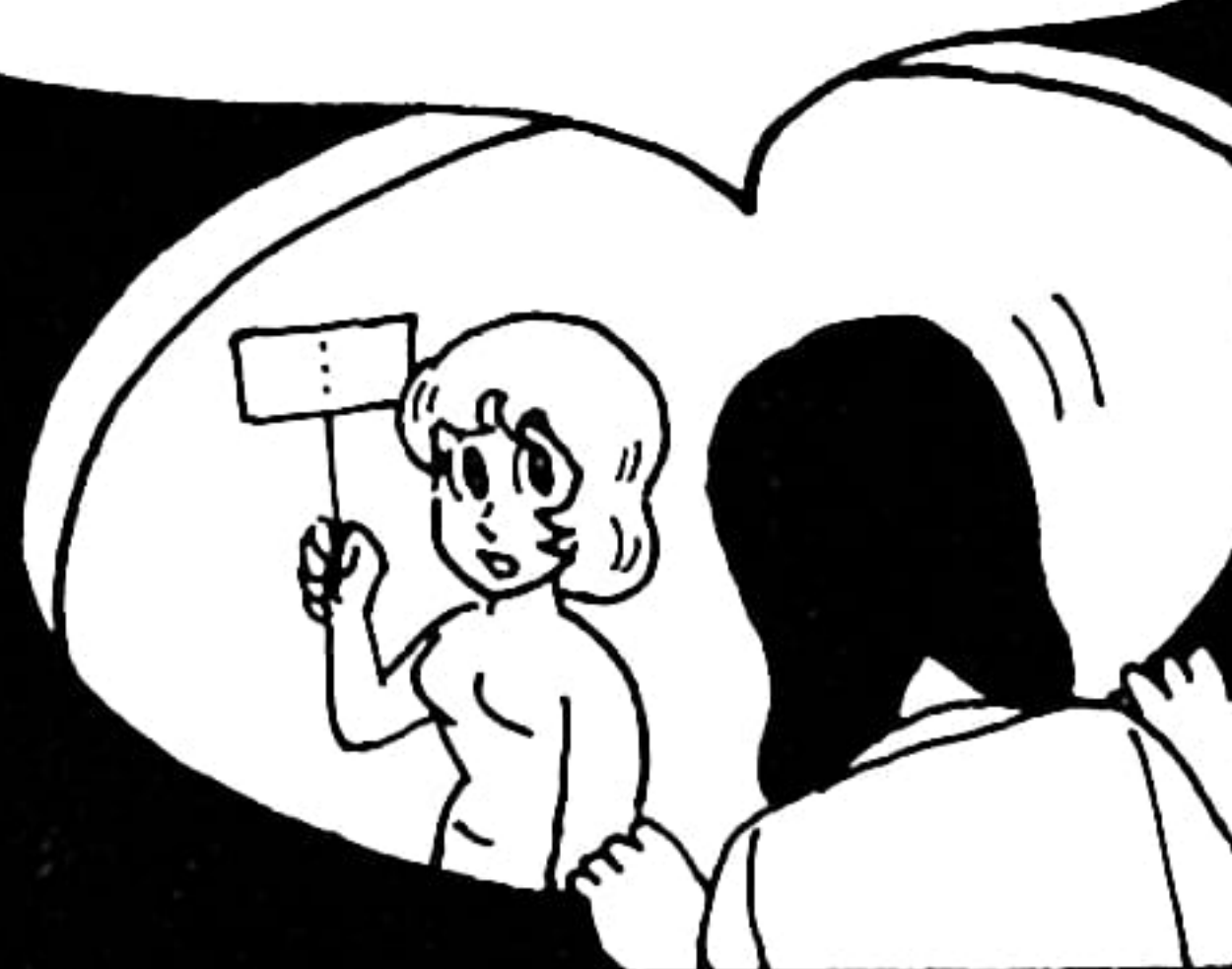
CPU
CD
?

このように、うまくデータをメモリの中にならべておくことを、プログラミングと言っているわけです。

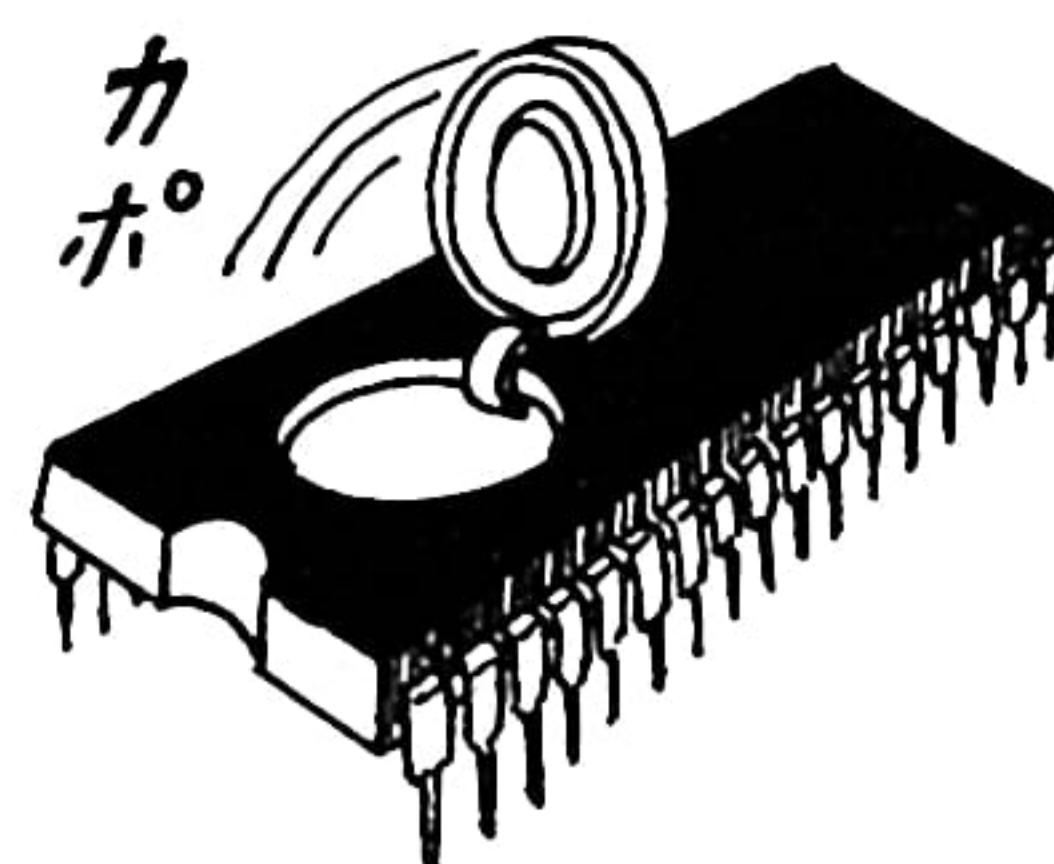
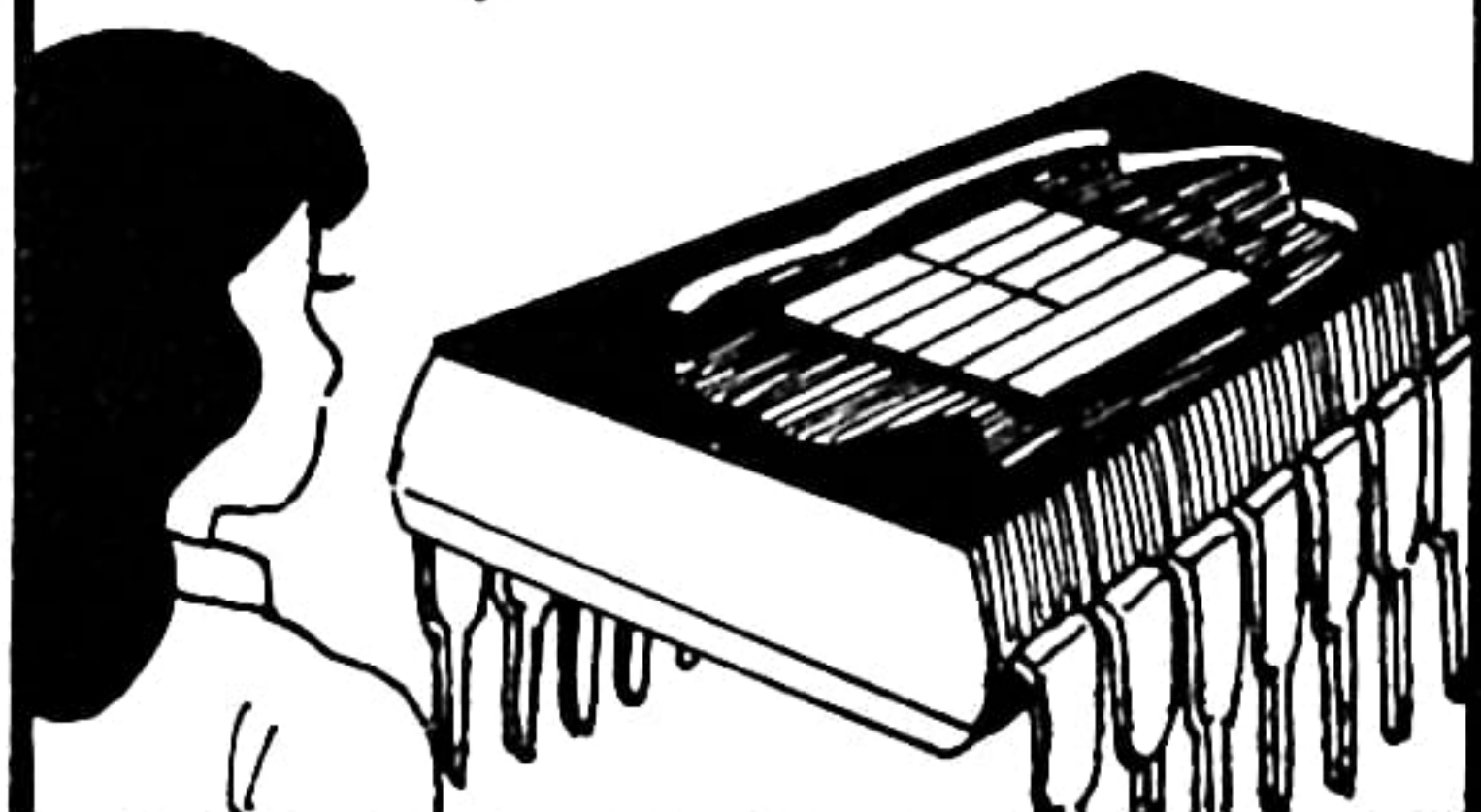
そこでメモリの中に入れるデータを工夫すれば、CPUがあたかも人工頭脳のような働きをするのです。

0F DA C2 02
0E 04 3E 90
C3 C6 02 ……

これらの働きや役目がチンプンカンプンでは、とうていマシン語のプログラムは組めません。

MVI␣C, 4

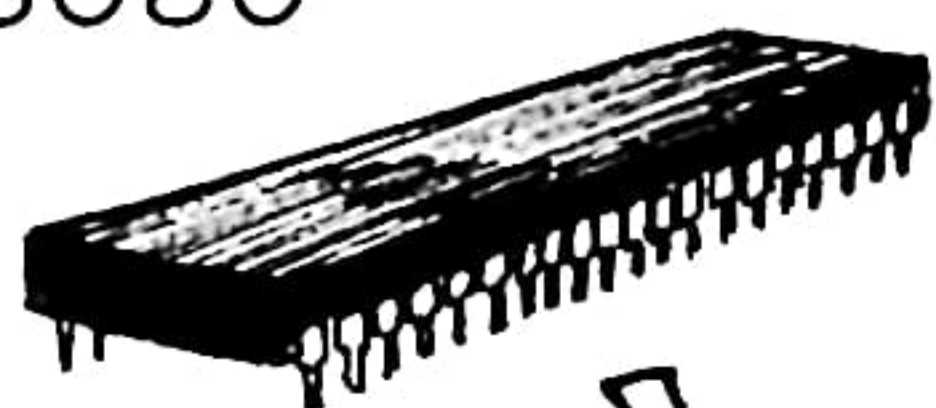
Cレジスタに4を入れる

たとえばレジスタ、プログラムカウンタ、スタックポインタ、フラグなどはCPUの中にあるものです。

また、CPUの中にあって、プログラムする上でどうしても知っておかなくてはならないこともあります。

カポ

そこでもう少しマンガチックに表現してみました。CPUの中にあってプログラムに関係があるものです。

8ビット	8ビット
A	F/F
B	C
D	E
H	L
PC	
SP	

16ビット

マニュアルなどにはこのように書いてありますがチンプンカンプンもいいところ……。

8bitマイクロコンピュータ
8080
CPUの中をのぞいてみると…。
レジスタたちが、CPUの中のどこにいるのか知っておくことはソフト的な機能を知るうえでとても参考になります。
アキュムレータ（Aレジスタ）
こりゃどーも
ややや
Aちゃんあそびましょ。
せーの
フラグ フリップフロップ
電源 +12V +5V −5V GND
Aレジスタ
Aラッチ
テンポラリレジスタ
フラグF/F
演算部 ALU
10進補正
D7…D0 双方向性バス
データバス バッファ/ラッチ
命令レジスタ
命令デコーダとマシンサイクルエンコーダ
8ビット内部データバス
データバス制御
割り込み制御
ホールド制御
WAIT制御
タイミングおよび制御部
SYNC クロック
WR DBIN 割込み許可 割込み要求 ホールド承認 ホールド WAIT READY SYNC φ1 φ2
リセット
マルチプレクサ
W テンポラリレジスタ
Z テンポラリレジスタ
レジスタ選択
B C D E H L
SP（スタックポインタ）
PC（プログラムカウンタ）
インクレメンタ/デクレメンタ およびアドレスラッチ
アドレスバッファ
A15…A0 アドレスバス
私たちはアツアツカップル。くっついたり、はなれたり…。
汎用レジスタ
スタック ポインタ
私は、データの保護が仕事です。
私は、プログラムの進行係です。
JUMP
プログラムカウンタ

⑤マシン語のしくみ

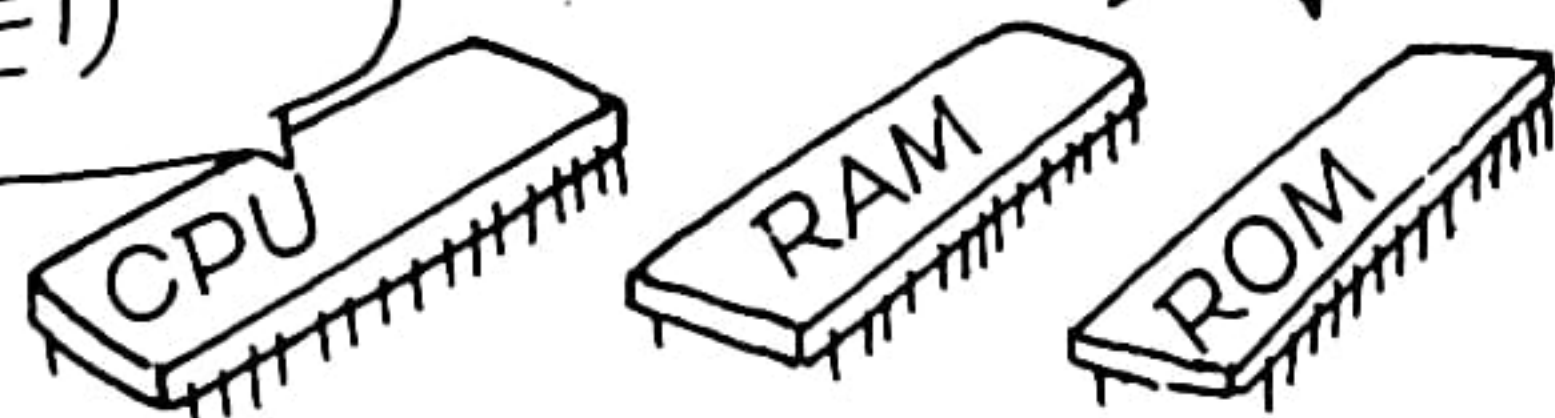
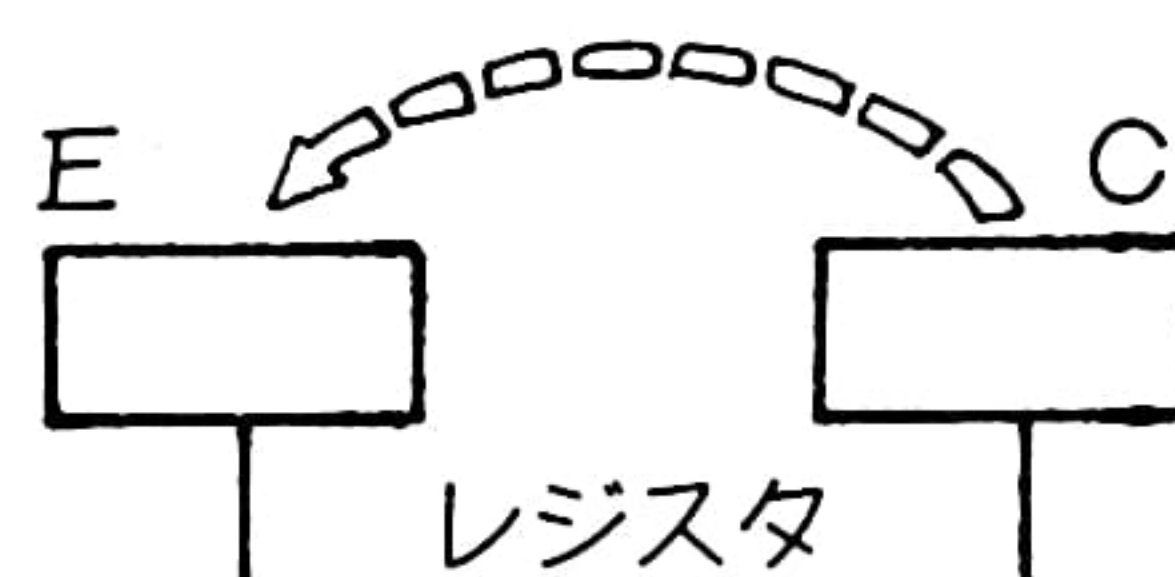
レジスタというのはCPUの中で一時的にデータを記憶する場所のことです。データとして8ビット、アドレスとして16ビット記憶できるように工夫されています。

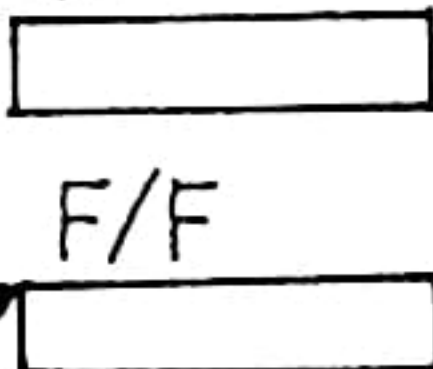
レジスタはCPUの内部メモリ

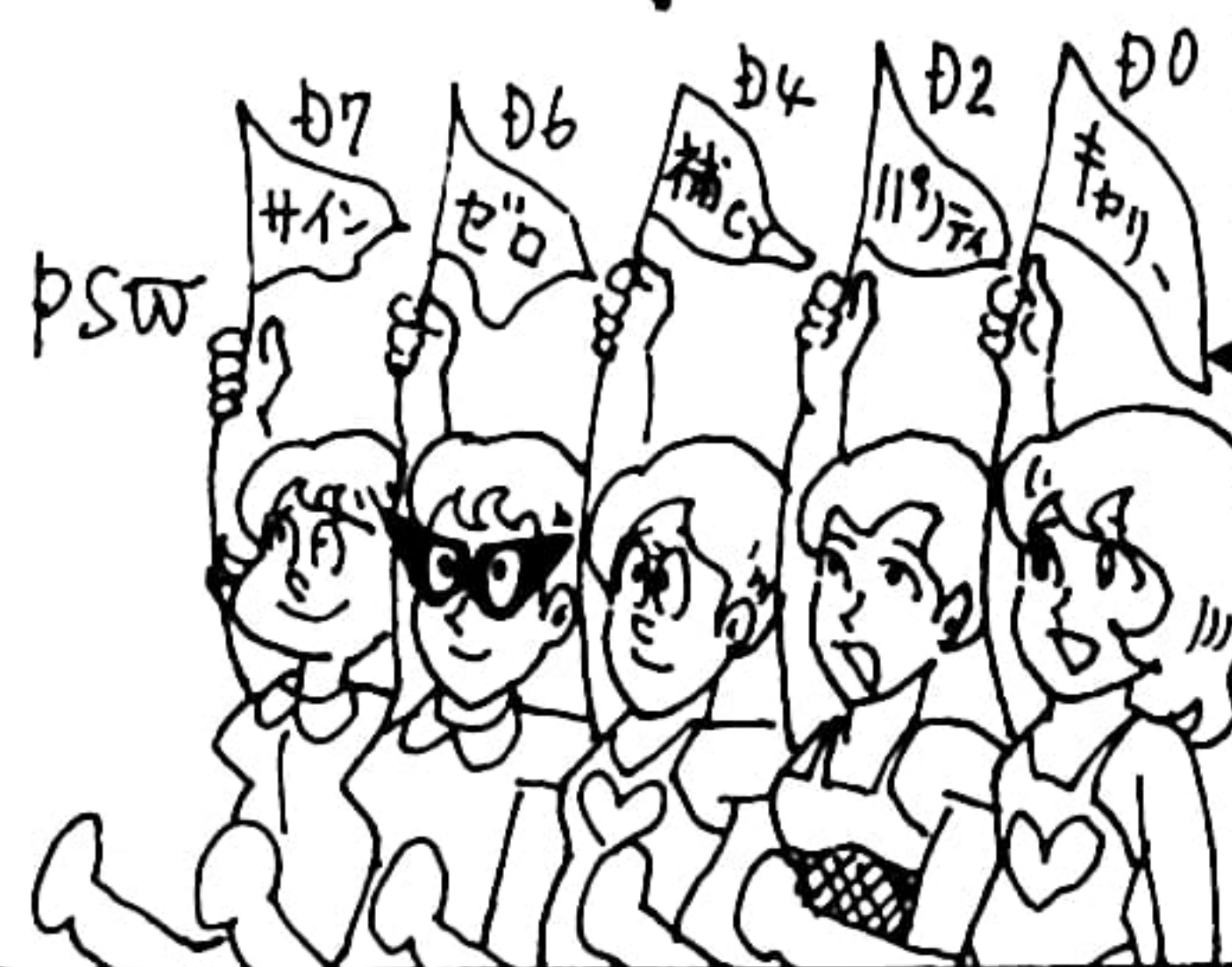
メモリ

CPU
RAM
ROM

MOV␣E, C

E
C
レジスタ

このアキュムレータとフラグの関係は、とてもコンピュータ的です。その話はまたあとでね。

AまたはAcc
F/F

私たちフリップフロップ(F/F)は、フラグビットの集まりです。1ビットごとに意味があり、プログラムで使います。

PSW
D7 サイン
D6 ゼロ
D4 補C
D2 パリティ
D0 キャリー

僕はアムキュレータ。8ビットのレジスタとしても使われますのでAレジスタとも呼ばれます。メモリのアドレスに相当するコードは3ビット111です。

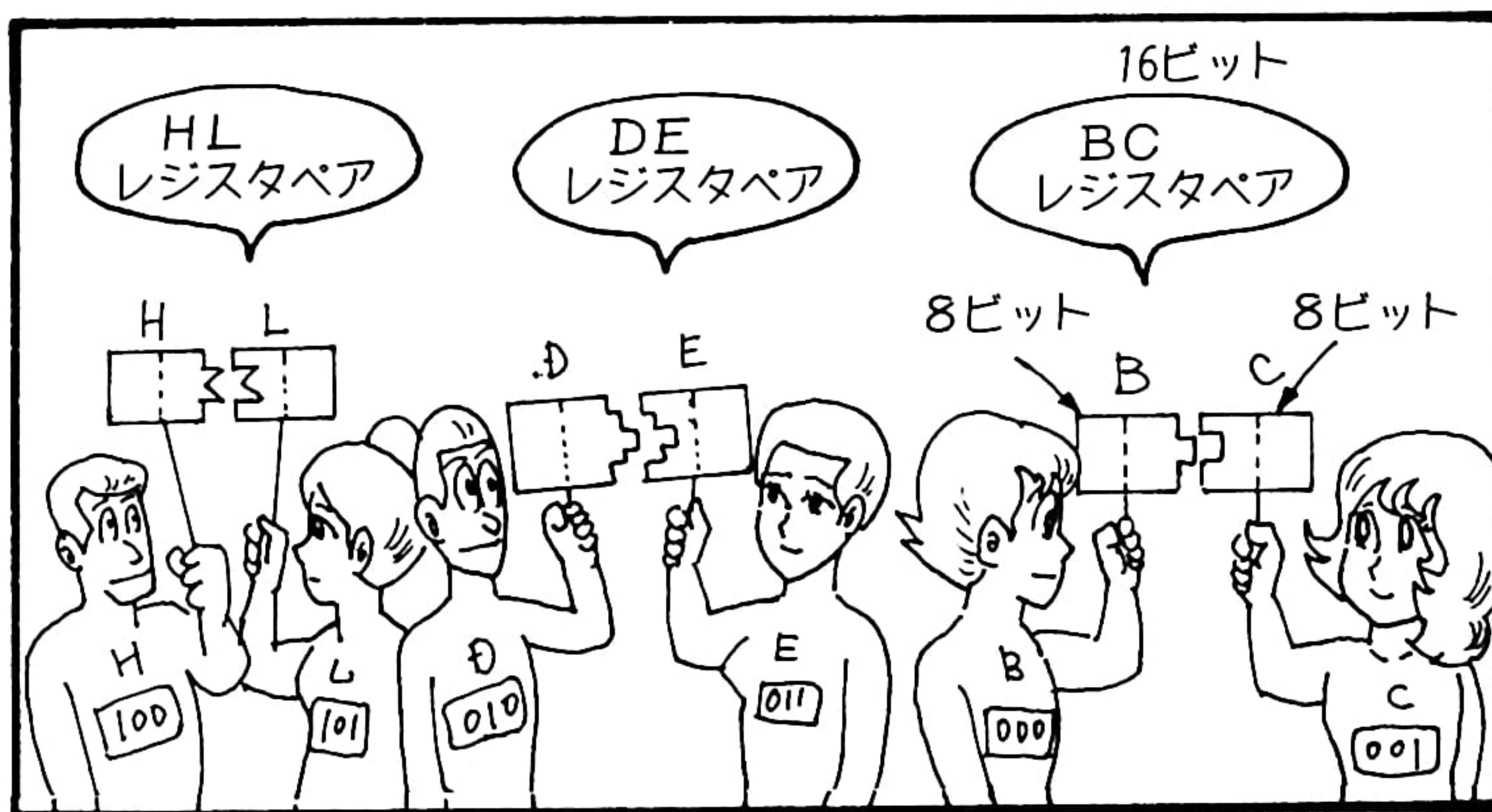
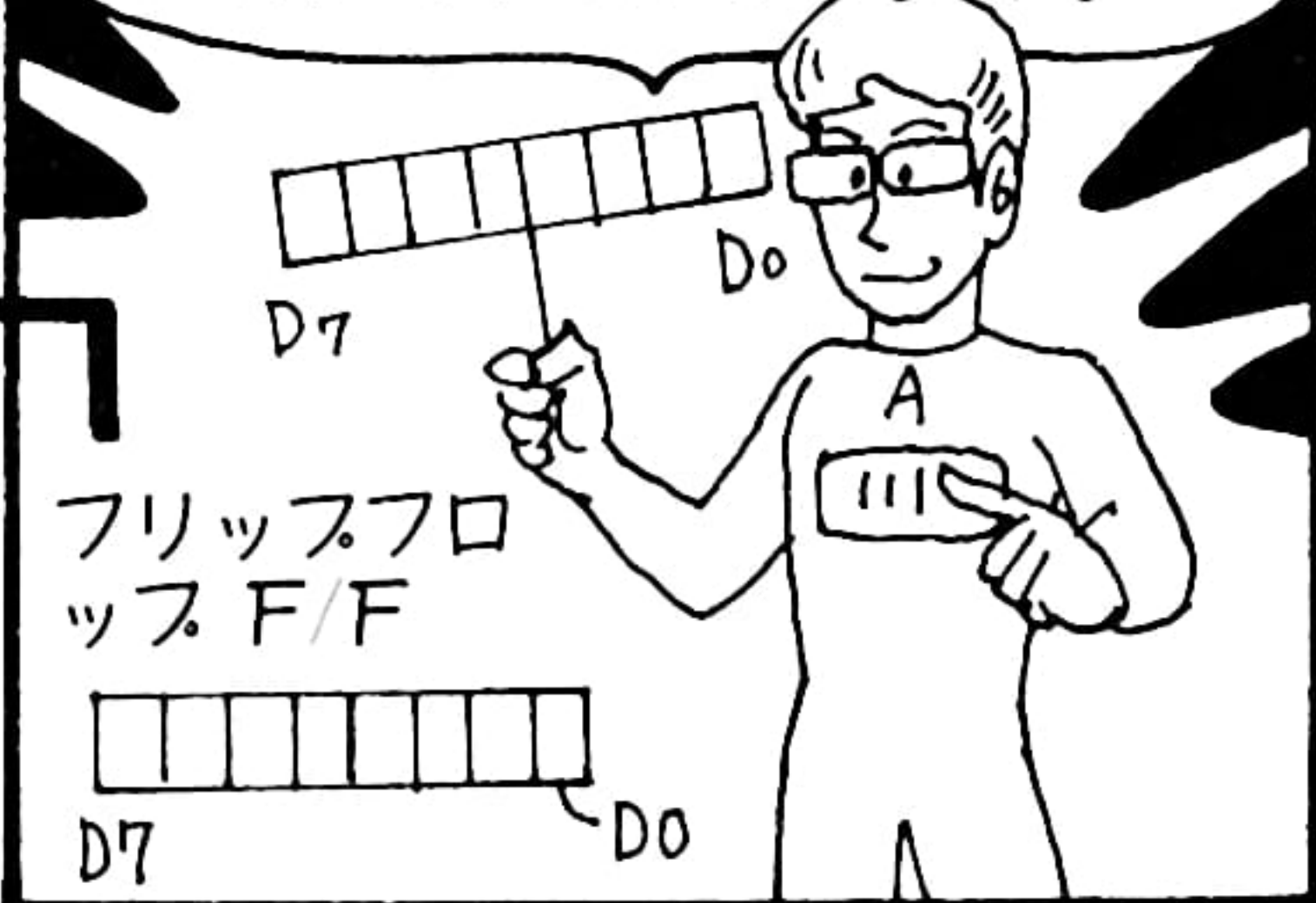
D7
D0
A
111
フリップフロップ F/F
D7
D0

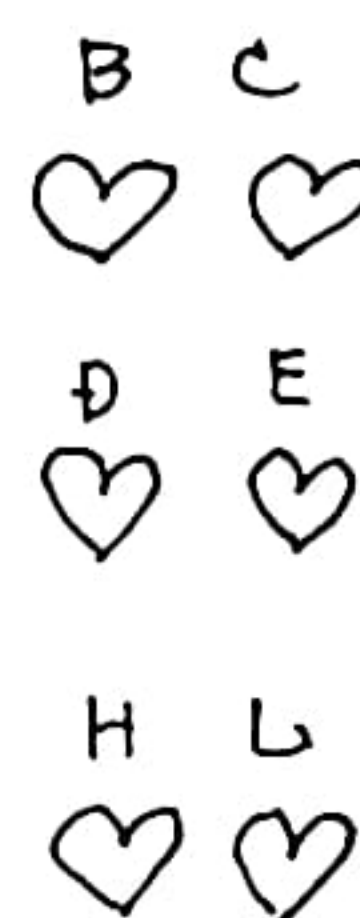
16ビット

HL レジスタペア

DE レジスタペア

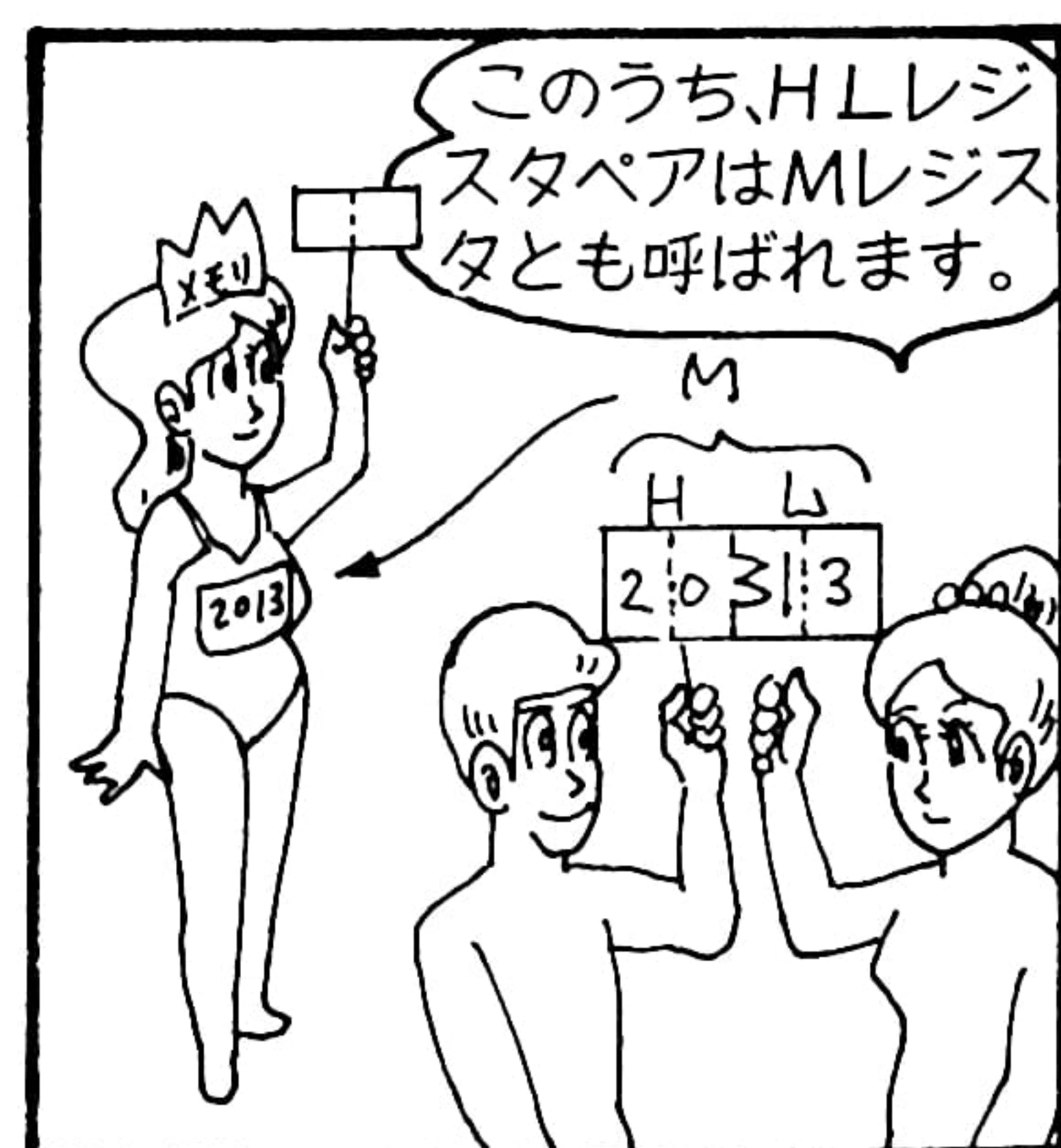
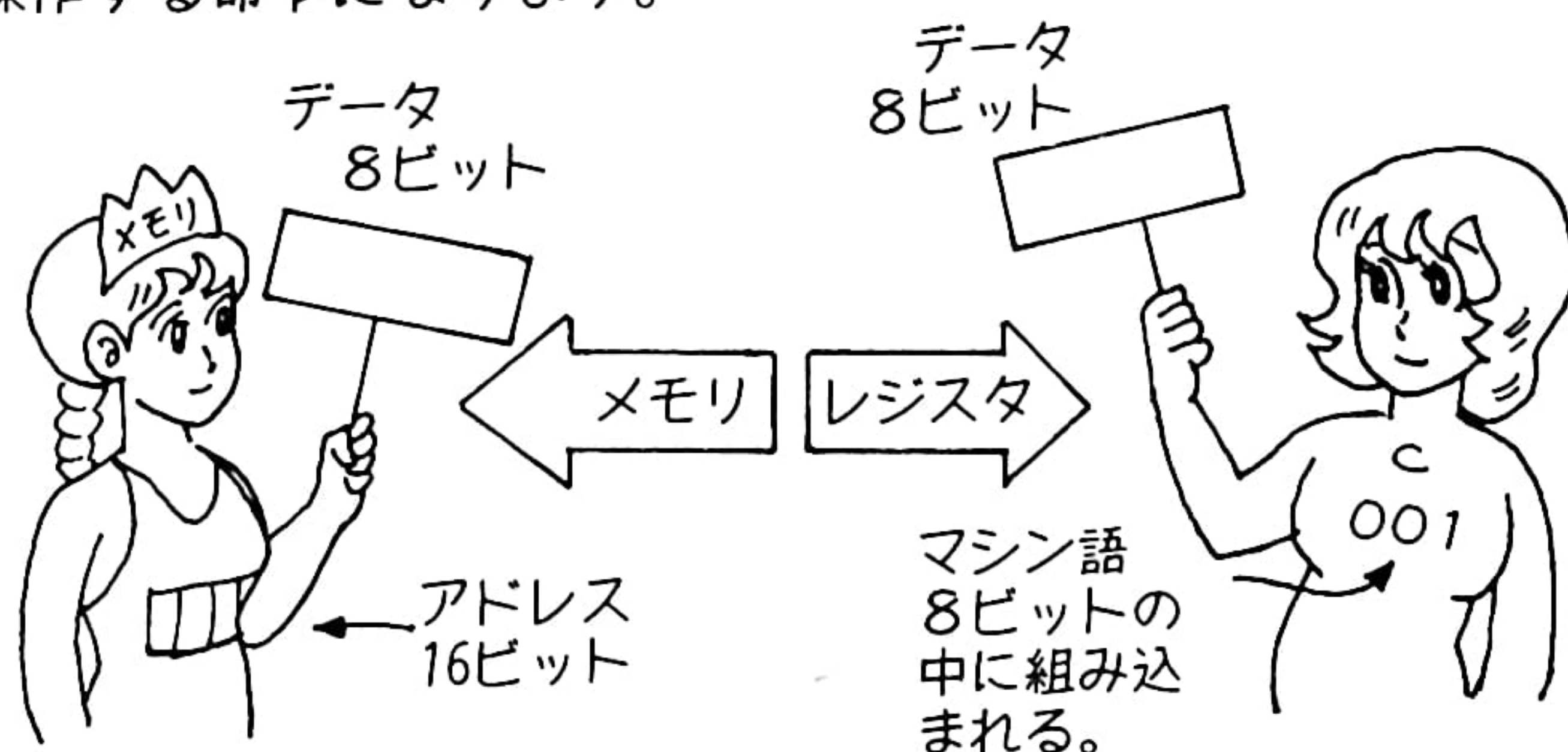
BC レジスタペア

H L
D E
8ビット
8ビット
B C
H 100
L 101
D 010
E 011
B 000
C 001

B、C、D、E、H、Lの6つのレジスタはそれぞれペアを組むことができます。

B C
D E
H L

レジスタにはROMやRAMのようにアドレスがありません。しかし、それぞれ3ビットのコードを持っていてこれがマシン語の中に組み込まれ、そのレジスタを操作する命令になります。

データ 8ビット
データ 8ビット
メモリ
メモリ
レジスタ
C 001
アドレス 16ビット
マシン語 8ビットの中に組み込まれる。

このうち、HLレジスタペアはMレジスタとも呼ばれます。

メモリ
2013
M
H L
2013

たとえば、レジスタ間のデータをやりとりする(転送という)命令は下のような形式で行なうように決められています。

この01がレジスタ間データ転送を意味する。

8ビットをこのように意味づける。

01 DDD SSS

データを受け取る側のレジスタコードを入れる。

データを送る側のレジスタのコードを入れる。

8080の場合、マシン語のコードは00〜FF(256通りある)で表現されますが適当に意味をつけているわけではありません。

00……何もせず次に進む
01……Bレジスタに2バイト入れる
02……(B)番地のデータをAレジスタに入れる
03……(BC)をインクリメント

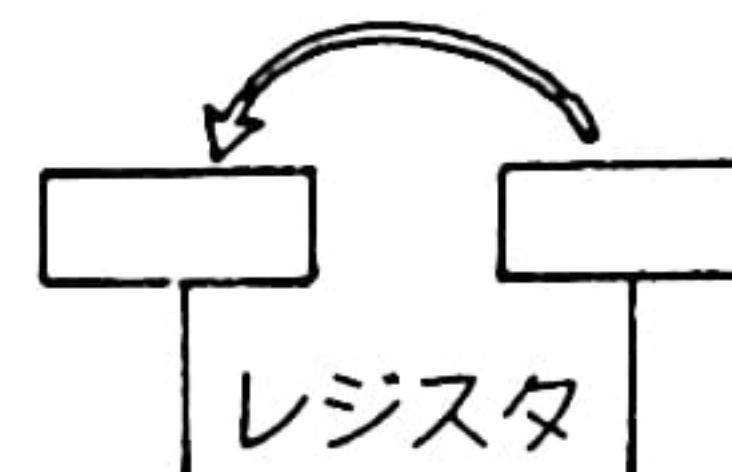
もちろん01001111なんていちいち書いていられないので、4ビットずつ2つに区切って4F(ヨンエフ)と表現します。

4 F
01001111

たとえばAレジスタの内容をCレジスタに送るマシン語はAのコード111とCのコード001を入れて、このようになります。

01 001 111
Cレジスタ Aレジスタ

だから、この01がレジスタ間データ転送であることを覚えているだけで、マシン語を作れるのです。

01………
レジスタ

今Aレジスタに47というデータが入っていたとして4Fというマシン語の命令が実行されると、Cレジスタに47が入ります。私たちはこの動きを4Fというコードから読みとることができません。

47
47
A 111
47が入り前のデータは消える
C 001
この時の動作
プログラム
4F

マニュアルやマイコン雑誌でみるマシン語はすべてこのように1バイト(8ビット)のデータを2つに区切って16進数で表現したものです。

CD3F3E41
3FFF5D

8080では、この場合、MOVがニーモニックになります。

スペース
コンマ
MOV␣C,A=4F
受ける側のレジスタ名
送る側のレジスタ名

そこでこのマシン語の意味を表現できるように考えたのがニーモニックコードなのです。

英数字表現
16進表現
ニーモニックと言う

つまりマシン語をつくることはできてもあとでそのコードを見て動作内容を思い出せるかどうかが問題です。

?
4F
えーと、えーと。忘却とは…。

では、MOV␣L,Cという命令の動作とマシン語はどうなるか見当がつきますか？

MOV␣L、C

動作はCレジスタの内容をLレジスタに転送することです。

L
101

C
001

マシン語はLレジスタのコードが101ですから、さっきと同様にして作れます。69（ロクキューです）。

6 9
01 101 001
L C

レジスタ間のデータ転送命令はこの調子でどんどん作れます。

MOV␣B,B
MOV␣C,B
MOV␣D,B
⋮
MOV␣B,C
MOV␣C,C
⋮

（ ）内はレジスタコード

A(111)	―
B(000)	C(001)
D(010)	E(011)
H(100)	L(101)

M(110)

次はCPUの外のメモリとレジスタ間のデータ転送を考えてみましょう。ハード的にはROMちゃんとRAMちゃんの２種類ありましたね。

RAM
ROM
CPU
RAM
ROM

まずROMちゃんですが、ROMちゃんはデータを送ることしかできません。ROMちゃんに書き込む命令を使っても無視されます。

ROM
CPU
3D

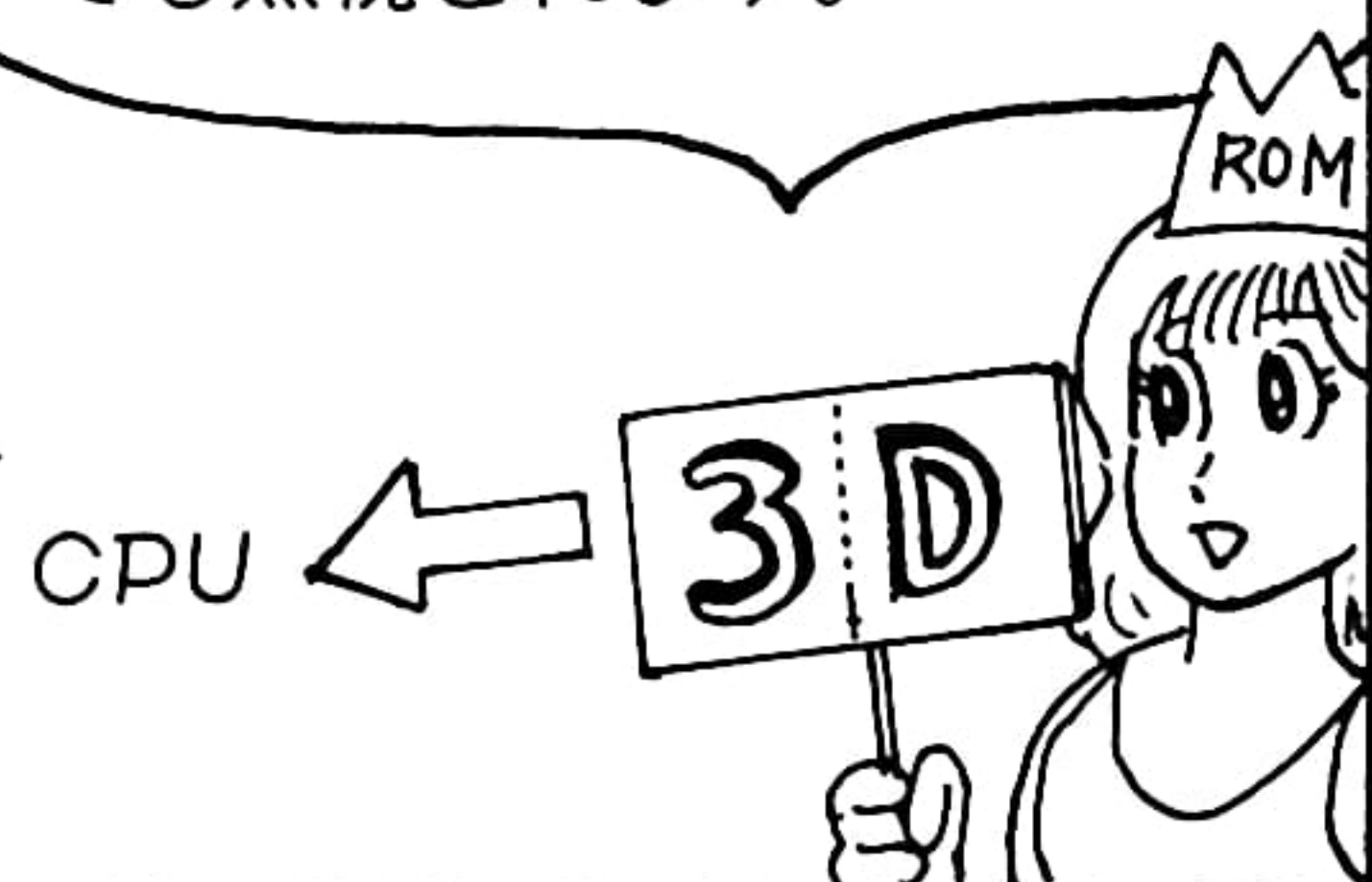
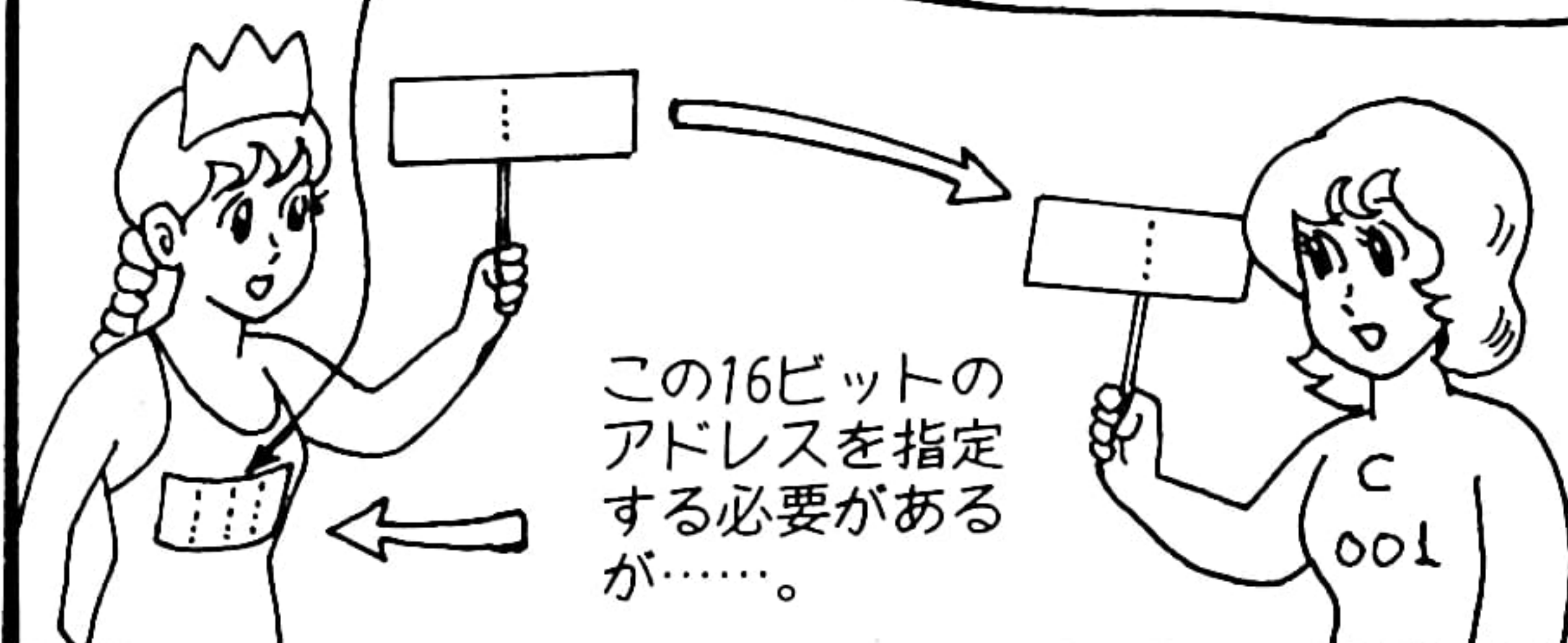
次はRAMちゃんですが、RAMちゃんはデータを送ることも受け取ることもできます。

RAM
CPU
FF
F002

どちらもアドレスがあらかじめハード設計の時に決められています。

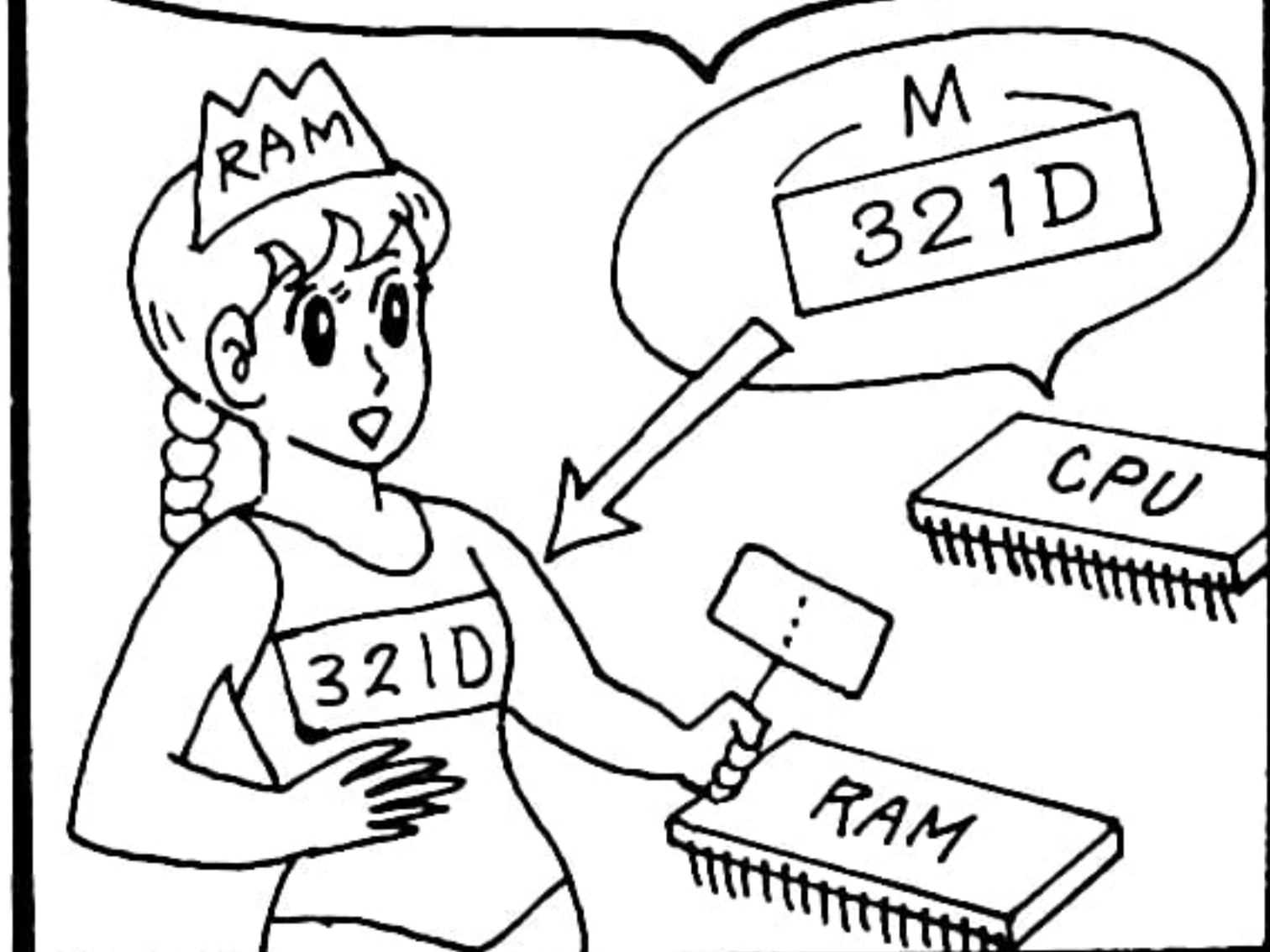
中身
RAM
ROM

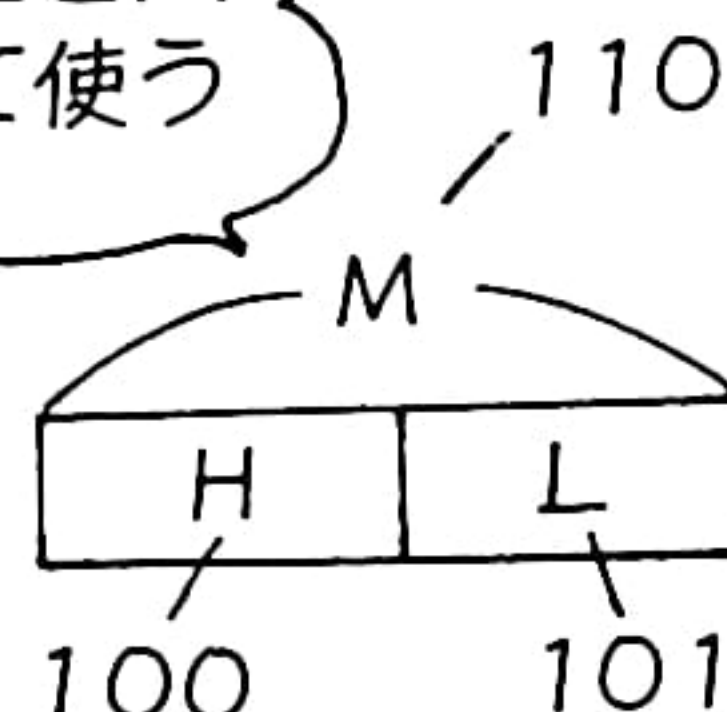
F000	2C
F001	3D
F002	FF
F003	AB
F004	53
F005	4C
F006	B5

レジスタはA、B、C、D、E、H、Lだけでしたがメモリは0000～FFFFまで、なんとRAMちゃん65536人分使えるのです（ただし、メモリが実装されていたらのお話ですが……）。

たとえば外のメモリからレジスタにデータを入れたい時はアドレス○○○○番と指定する必要があります。

この16ビットのアドレスを指定する必要があるが……。

C
001

このMつまりHLレジスタの内容がアドレスを示すのです。

M
321D
CPU
321D
RAM

さきほどお話した、HLレジスタペアにMという名前をつけてこのコードを110としたのです。

HとLをMとして使う
110
M
H L
100 101

実はこの16ビットのアドレス指定はとてもうまく考えられています。

?

ではMOV␣M,Cのマシン語を作ってみましょう。簡単ですね。71（ナナイチ）です。

7 1
01110001
M C

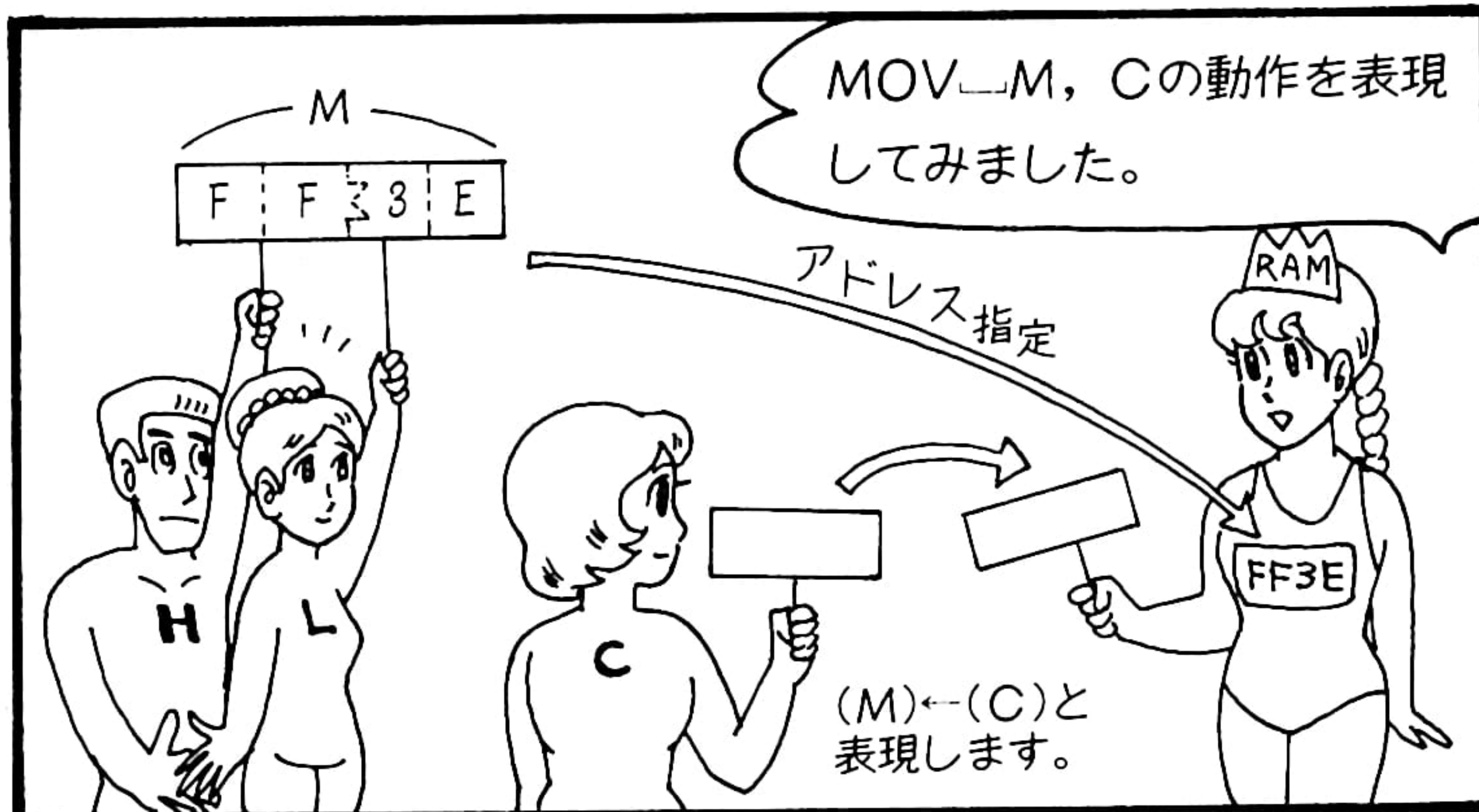
レジスタからメモリへのデータ転送のマシン語はこうなります。

01110SSS
M

メモリからレジスタへデータを送る時のマシン語はこうなり、

01SSS110
M

M
FF3E
H L C
アドレス指定
FF3E
(M)←(C)と表現します。

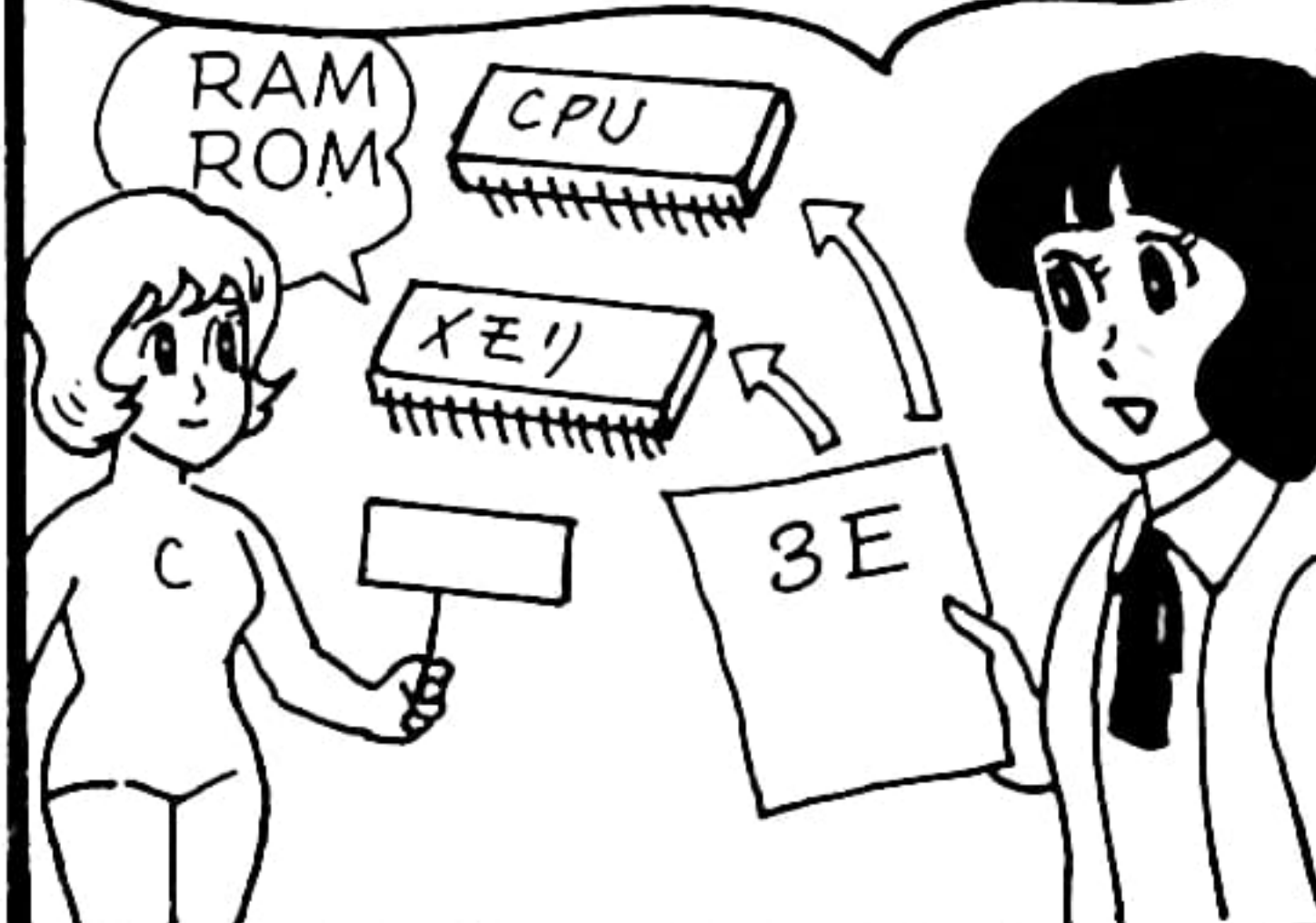
MOV␣M，Cの動作を表現してみました。

MOV␣C，Mはどうでしょうか。4E(ヨンイー)となりました。

4 E
01001110
C M

まず、1バイトのデータをレジスタに入れるためのマシン語です。DDDはA、B、C、D、E、H、L、Mのコードを入れますが,Mのときはレジスタでなくメモリにデータが入ることになります。

00DDD110

次は1バイトや2バイトのデータをソフト的に入れたい時のマシン語の作り方をお話しましょう。

RAM ROM
CPU
メモリ
C
3E

MOV␣A，Aというマシン語はあるのですが,MOV␣M,Mはありません。そのマシン語は76（ナナロク）となりますが,このコードはHLT(ホルト）といってCPUは停止してしまいます。

7 6
01110110
M M

ここでひとつの疑問が生じます。このOEと3Eの区別をどうつけるのかということです。つまりこの3EはCPUに動作をさせたいマシン語ではないのです。

3Eってマシン語もあるわけでしょう?

このマシン語はOE3Eと2バイトになってしまいました。

MVI␣C,3E

OE
3E

ニーモニックはMVIに変わります。MVI␣C，3Eのマシン語はOE、その後にデータの3Eをつけます。

O E
00001110
C

つまり、モニタで使っていたOE3E4E0078…といったものの中には、CPUに動作させるためのコードと、そうでない定数があることがわかります。

これを命令コードとして読めば
これがデータとして読まれる
31C8203E0106

つまりすぐあとの1バイトである3Eは定数として読み込まれ、次にCPUがマシン語として読むのは定数のあとの次の1バイトです。

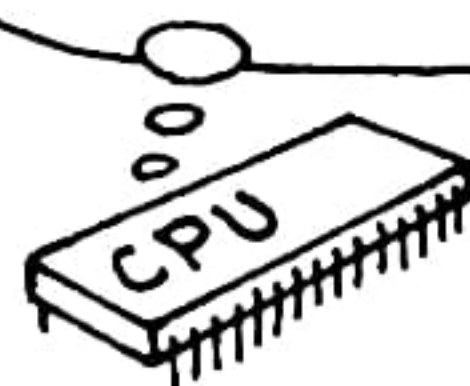
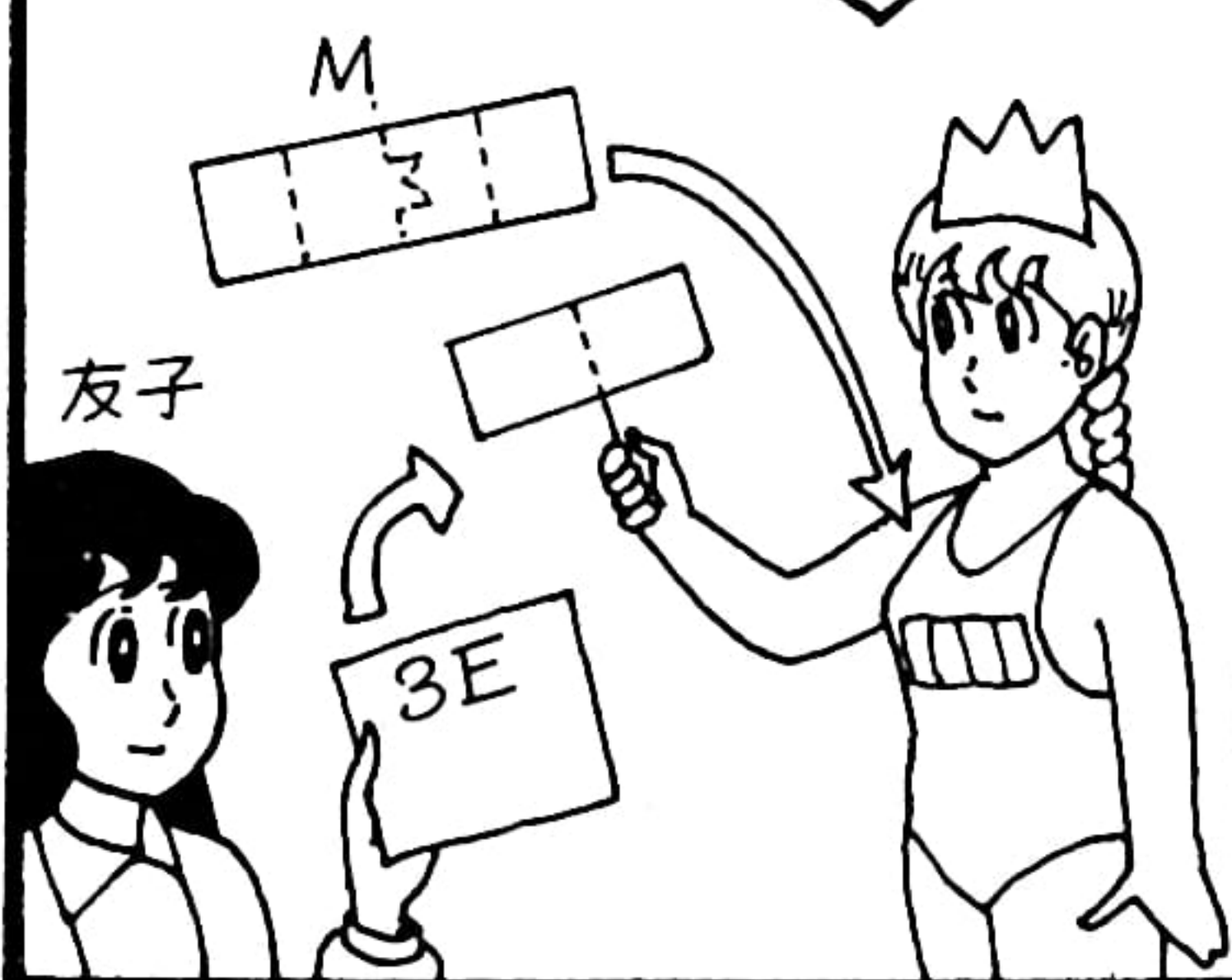
OE
3E データ
31 → これが次の命令コードになります。
下位アドレス

ところが、そこはうまくできています。CPUはOEを読むと次の1バイトが定数であると判断するようになっているのです。

おや、OEが来た。ではもう1バイト読み込もう。

CPU
OE
3E

MVI␣M，3Eの動作です。

M
3
友子
3E

次にMVI␣M，3Eのオペコードをつくってみましょう。36（サンロク）となりました。

3 6
001101103E
M

今までマシン語と言ってきましたが、定数と区別するためオペコード（オペレーションコード）と言い換えることにしましょう。

MVI␣D，FF
16FF
オペコード データ

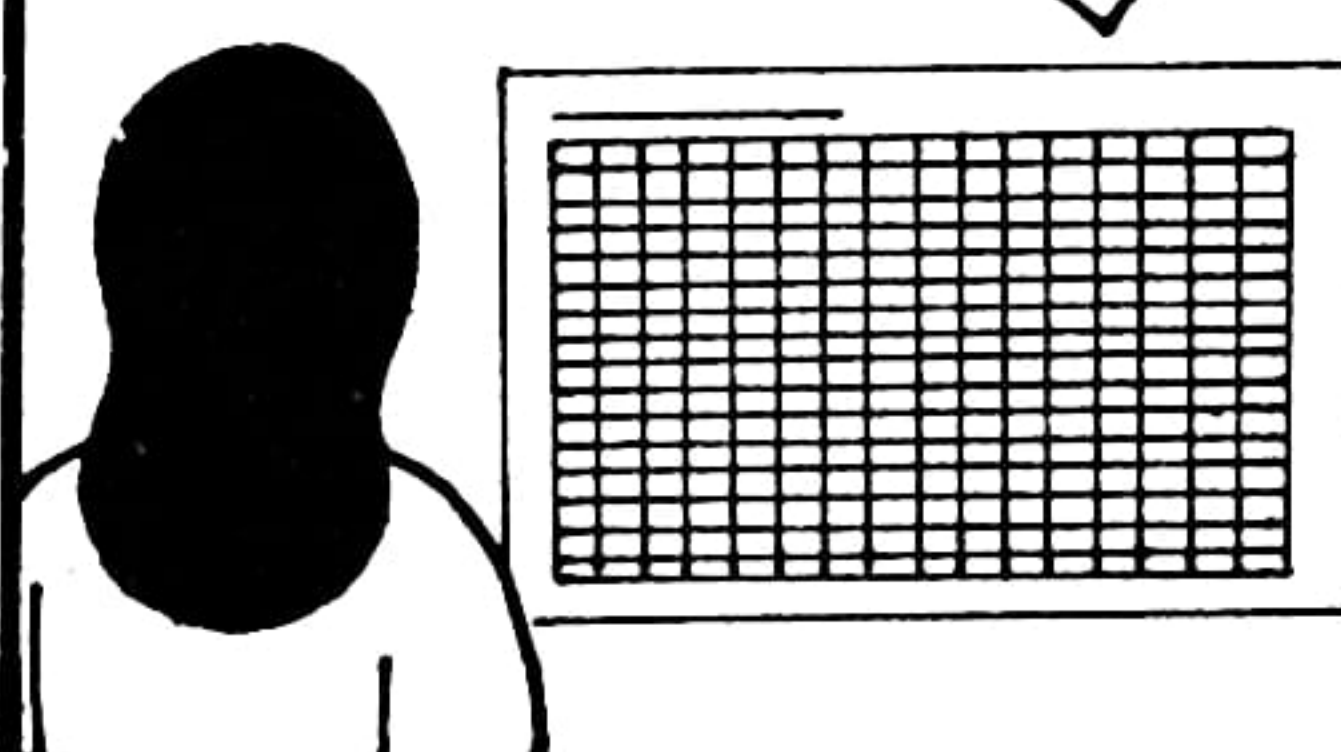
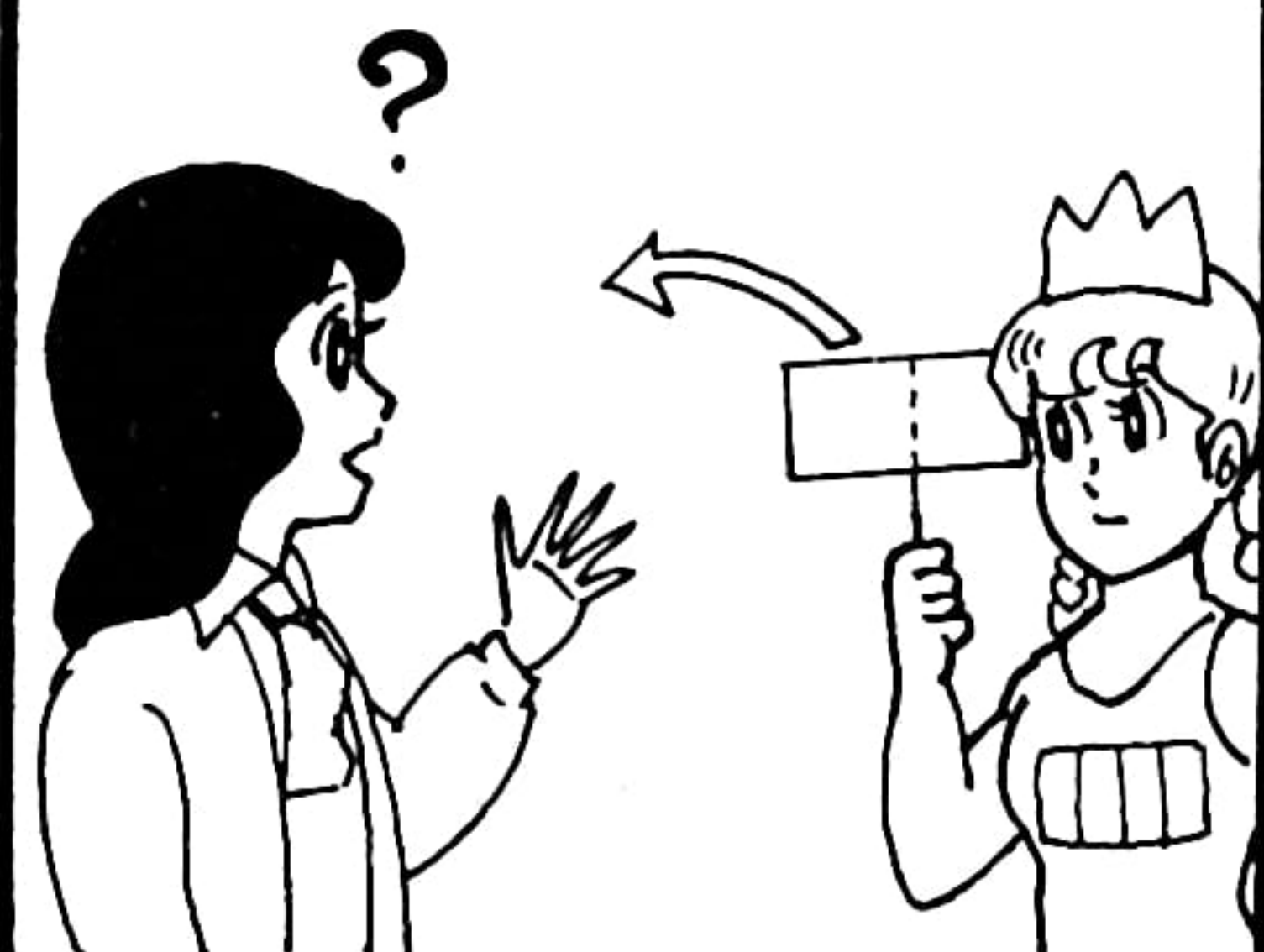
なおこれは8080用にインテル社が考えたニーモニックです。この表はアセンブラ記述形式で書いてありますのでとても役に立つものです。

MOV、MVI、LXIなどが出てきたらこれはインテルのニーモニックということね。

次のページにニーモニックとオペコードの対照表を示します。今作ったオペコードがあるか確かめてみてください。

MVI␣3E，Mはありません。

?

⑥8080コード表

8080CPUのニーモニック-OPコード割当表

(Z80)はZ80CPUのOPコードに使われます。
D8は1バイトのデータ、D16は2バイトのデータを示します。Adrはアドレス2バイトです。

␣はスペースを示します。
アセンブラを使用する時便利なようにMOV␣D,Bのようにオペランド部を含めて書いています。

下桁＼上桁	0・	1・	2・	3・	4・	5・	6・	7・	8・	9・	A・	B・	C・	D・	E・	F・
・0	NOP	NOP (Z80)	NOP (Z80)	NOP (Z80)	MOV␣B,B	MOV␣D,B	MOV␣H,B	MOV␣M,B	ADD␣B	SUB␣B	ANA␣B	ORA␣B	RNZ	RNC	RPO	RP
・1	LXI␣B,D16	LXI␣D,D16	LXI␣H,D16	LXI␣SP,D16	MOV␣B,C	MOV␣D,C	MOV␣H,C	MOV␣M,C	ADD␣C	SUB␣C	ANA␣C	ORA␣C	POP␣B	POP␣D	POP␣H	POP␣PSW
・2	STAX␣B	STAX␣D	SHLD␣Adr	STA␣Adr	MOV␣B,D	MOV␣D,D	MOV␣H,D	MOV␣M,D	ADD␣D	SUB␣D	ANA␣D	ORA␣D	JNZ␣Adr	JNC␣Adr	JPO␣Adr	JP␣Adr
・3	INX␣B	INX␣D	INX␣H	INX␣SP	MOV␣B,E	MOV␣D,E	MOV␣H,E	MOV␣M,E	ADD␣E	SUB␣E	ANA␣E	ORA␣E	JMP␣Adr	OUT␣D8	XTHL	DI
・4	INR␣B	INR␣D	INR␣H	INR␣M	MOV␣B,H	MOV␣D,H	MOV␣H,H	MOV␣M,H	ADD␣H	SUB␣H	ANA␣H	ORA␣H	CNZ␣Adr	CNC␣Adr	CPO␣Adr	CP␣Adr
・5	DCR␣B	DCR␣D	DCR␣H	DCR␣M	MOV␣B,L	MOV␣D,L	MOV␣H,L	MOV␣M,L	ADD␣L	SUB␣L	ANA␣L	ORA␣L	PUSH␣B	PUSH␣D	PUSH␣H	PUSH␣PSW
・6	MVI␣B,D8	MVI␣D,D8	MVI␣H,D8	MVI␣M,D8	MOV␣B,M	MOV␣D,M	MOV␣H,M	HLT	ADD␣M	SUB␣M	ANA␣M	ORA␣M	ADI␣D8	SUI␣D8	ANI␣D8	ORI␣D8
・7	RLC	RAL	DAA	STC	MOV␣B,A	MOV␣D,A	MOV␣H,A	MOV␣M,A	ADD␣A	SUB␣A	ANA␣A	ORA␣A	RST␣0	RST␣2	RST␣4	RST␣8
													このコードはハード的にデータバスに入れる			
・8	NOP (Z80)	NOP (Z80)	NOP (Z80)	NOP (Z80)	MOV␣C,B	MOV␣E,B	MOV␣L,B	MOV␣A,B	ADC␣B	SBB␣B	XRA␣B	CMP␣B	RZ	RC	RPE	RM
・9	DAD␣B	DAD␣D	DAD␣H	DAD␣SP	MOV␣C,C	MOV␣E,C	MOV␣L,C	MOV␣A,C	ADC␣C	SBB␣C	XRA␣C	CMP␣C	RET	RET (Z80)	PCHL	SPHL
・A	LDAX␣B	LDAX␣D	LHLD␣Adr	LDA␣Adr	MOV␣C,D	MOV␣E,D	MOV␣L,D	MOV␣A,D	ADC␣D	SBB␣D	XRA␣D	CMP␣D	JZ	JC␣Adr	JPE␣Adr	JM␣Adr
・B	DCX␣B	DCX␣D	DCX␣H	DCX␣SP	MOV␣C,E	MOV␣E,E	MOV␣L,E	MOV␣A,E	ADC␣E	SBB␣E	XRA␣E	CMP␣E	JMP (Z80)	IN␣D8	XCHG	EI
・C	INR␣C	INR␣E	INR␣L	INR␣A	MOV␣C,H	MOV␣E,H	MOV␣L,H	MOV␣A,H	ADC␣H	SBB␣H	XRA␣H	CMP␣H	CZ␣Adr	CC␣Adr	CPE␣Adr	CM␣Adr
・D	DCR␣C	DCR␣E	DCR␣L	DCR␣A	MOV␣C,L	MOV␣E,L	MOV␣L,L	MOV␣A,L	ADC␣L	SBB␣L	XRA␣L	CMP␣L	CALL␣Adr	CALL (Z80)	CALL (Z80)	CALL (Z80)
・E	MVI␣C,D8	MVI␣E,D8	MVI␣L,D8	MVI␣A,D8	MOV␣C,M	MOV␣E,M	MOV␣L,M	MOV␣A,M	ADC␣M	SBB␣M	XRA␣M	CMP␣M	ACI␣D8	SBI␣D8	XRI␣D8	CPI␣D8
・F	RRC	RAR	CMA	CMC	MOV␣C,A	MOV␣E,A	MOV␣L,A	MOV␣A,A	ADC␣A	SBB␣A	XRA␣A	CMP␣A	RST␣1	RST␣3	RST␣5	RST␣7
													このコードはハード的にデータバスに入れる			

⑦マシン語
こんな命令もある

プログラム
8E
データ
(M) 97
おい次の命令は
8Eだ。
(A)←(A)+(M)
+CARRY
をやるぞ。
RAM
CPU
???
Aレジスタ
HLレジスタペア
キャリーフラグ
集まれ！　あっそうだ。
データ(M)は外のRAM
からもってこなきゃ。

急ぐ人は6ペー
ジ先にジャンプ
してくださいね。

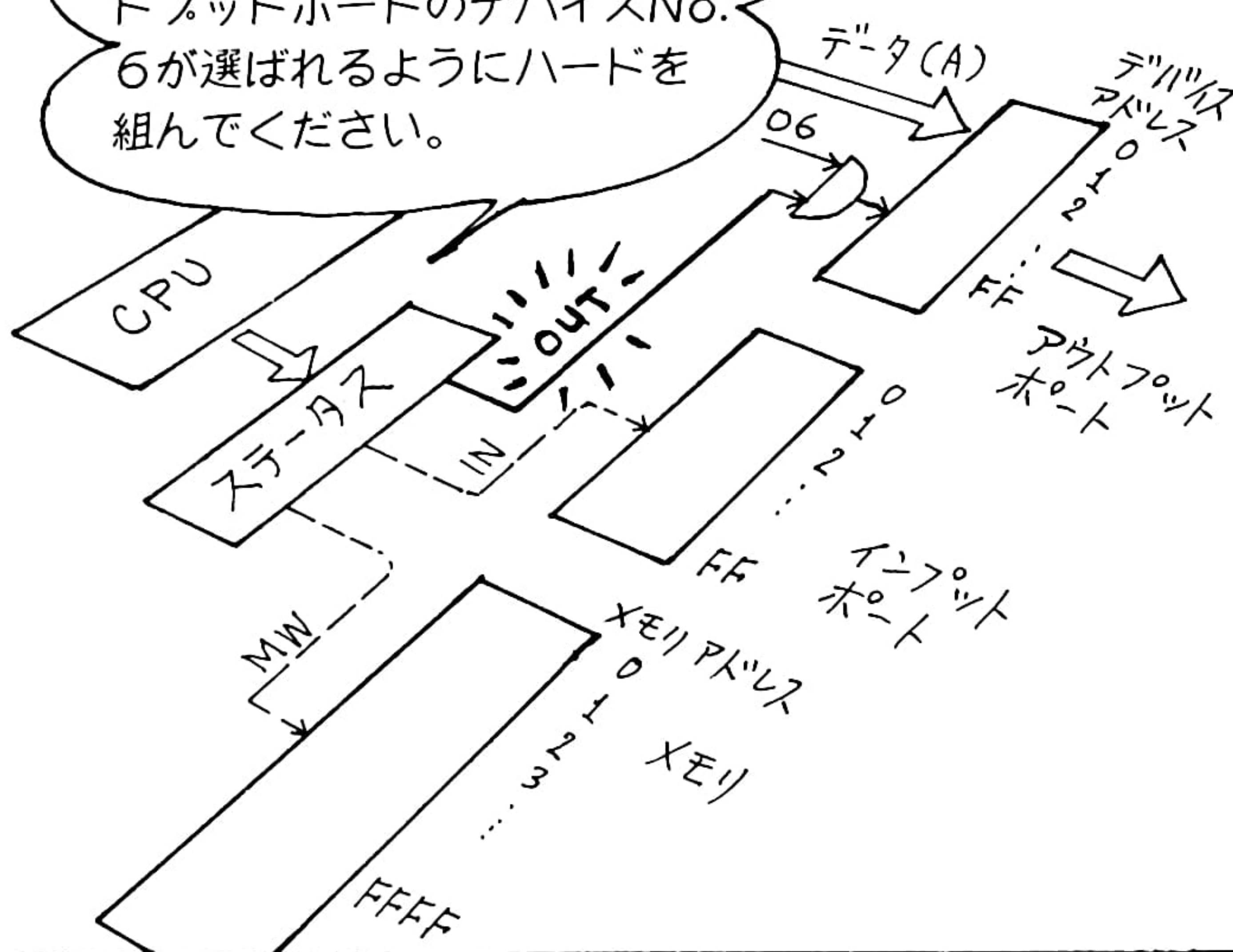
D7 D6 D5 D4 D3 D2 D1 D0
FLAGS OF 8080A
無視できないよ。このウーマンパワー
D7 サイン
D6 ゼロ
D4 補助キャリー
D2 パリティ
D0 キャリー

ADC M
8E　(これは１バイト命令だから
ONEパターン)
ビットに直してたし算する
のがコツです。答はもう一
度16進に。
AC
CY
与式
CY+(A)+(M)
=1+28+97
≠C0
0000 0001
0010 1000
+) 1001 0111
1100 0000
AC
ALU
いやんなっち
まうんだよな。
10進補正しな
いと答が合わ
ないんだ。
(A)
28
C0
16進の計算が
できる人には、
この答が合っ
ていることが
わかりますね。
(M)
97
RAM
メモリ
RAM
2003 97
2003
(M)⇐(H)(L)
H
L
20 03
(M) メモリのアドレス指定

ANI ␣ 38
〈例〉
E6 38 ⇨ 00111000
この命令は特定のビットを調べた
り、マスクしたりするのに用いら
れる。8bitのANDがソフトで実
現できるわけ
このAレジスタの内容
は、たとえばインプット
ポートからあらかじめ
読み込まれたもの
52
A
10
01010010
00010000
結果はA
レジスタ
に戻され
る。
フラグ

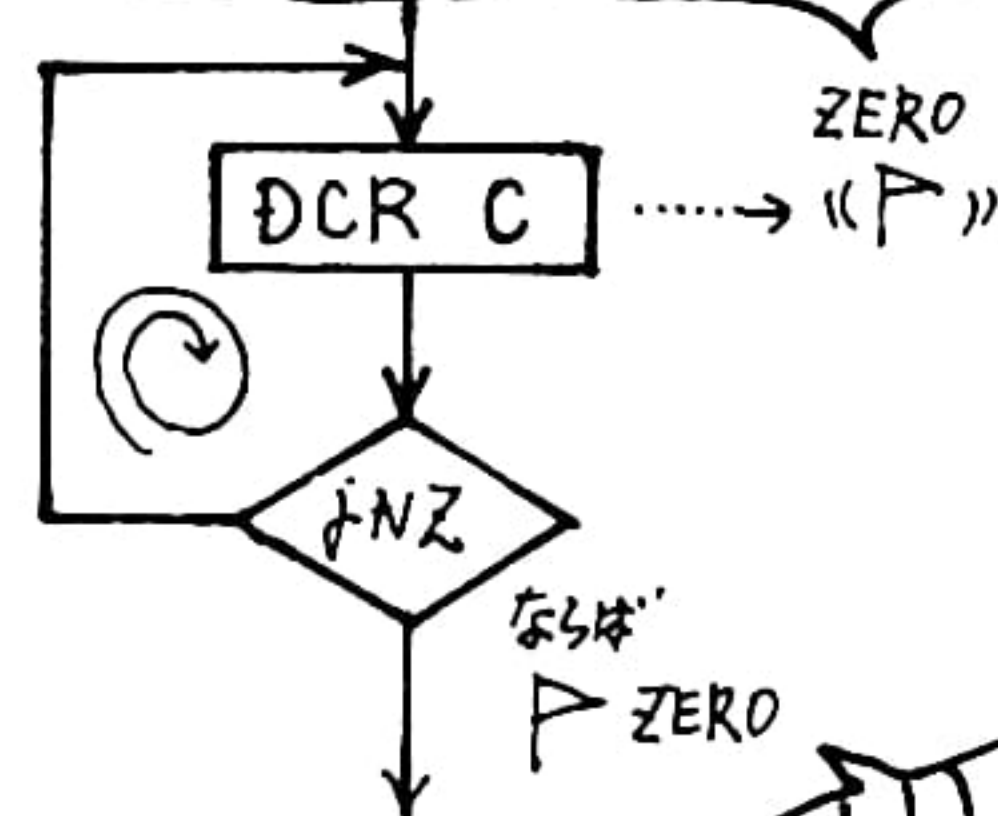
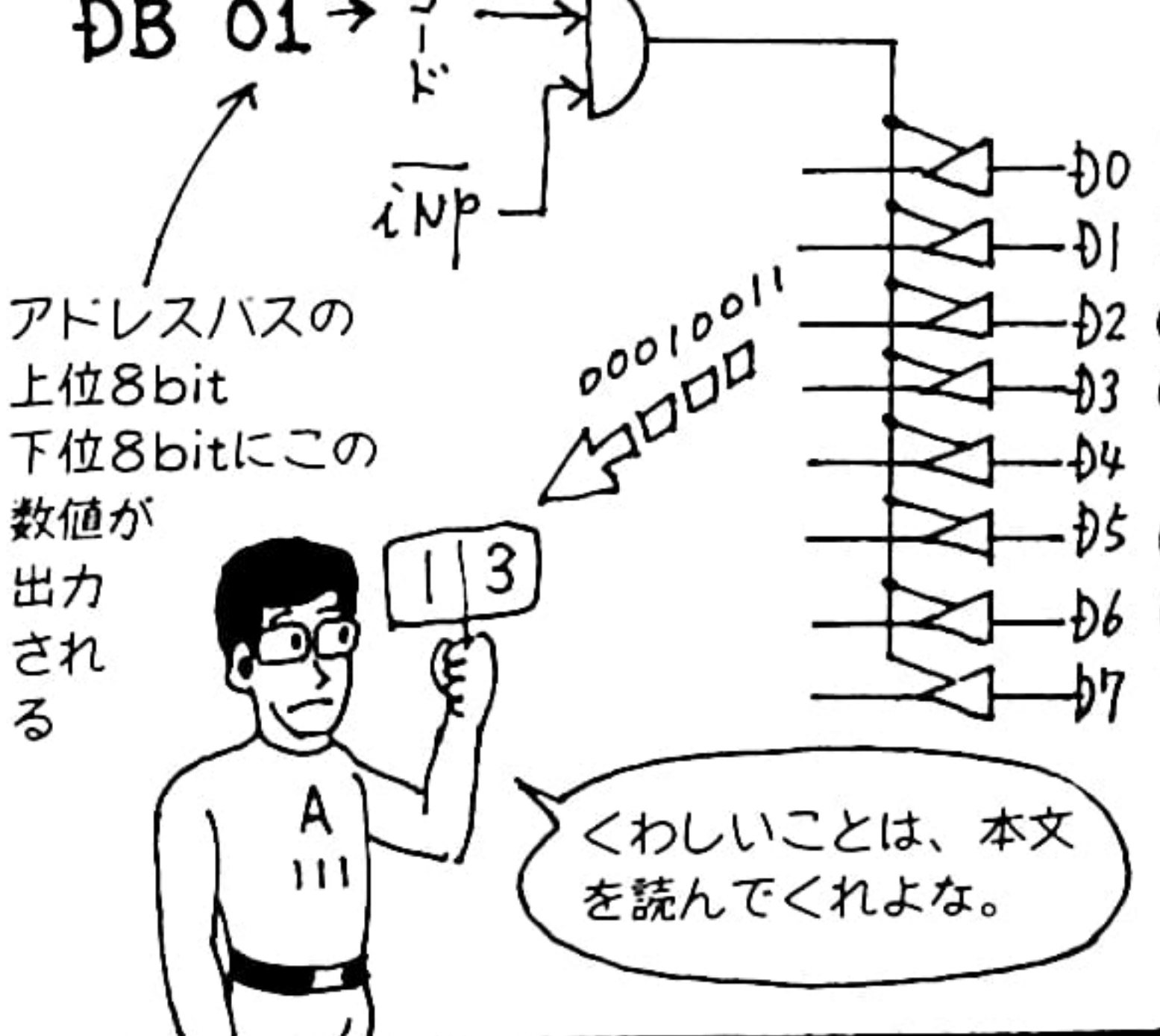
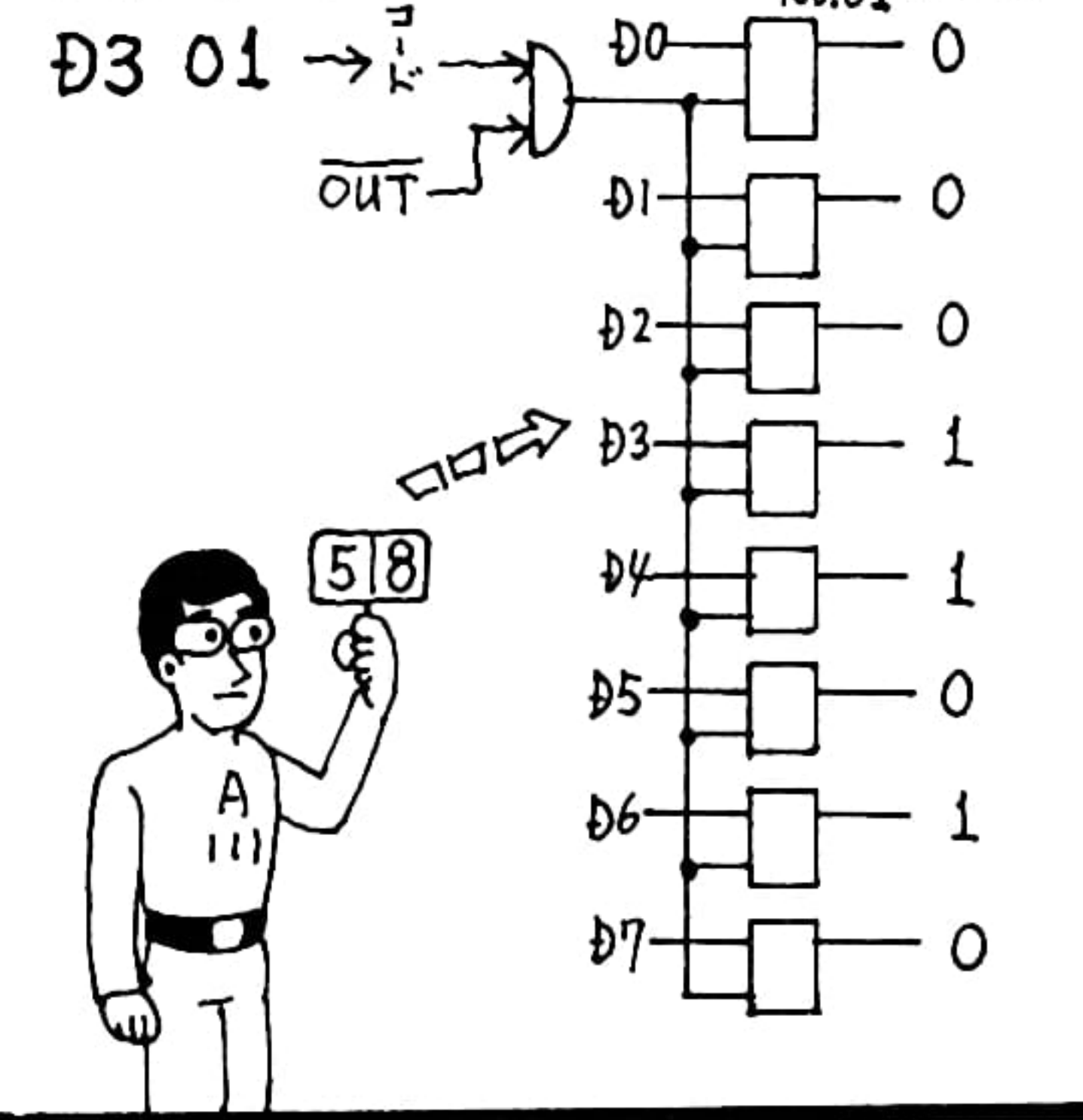
メモリの他にインプットポートとアウトプットポートがついているのが8080系CPUの特徴のひとつです。それぞれ256バイト分のデバイスNo.を指定できます。

OUT␣06を実行するとアウトプットポートのデバイスNo.6が選ばれるようにハードを組んでください。

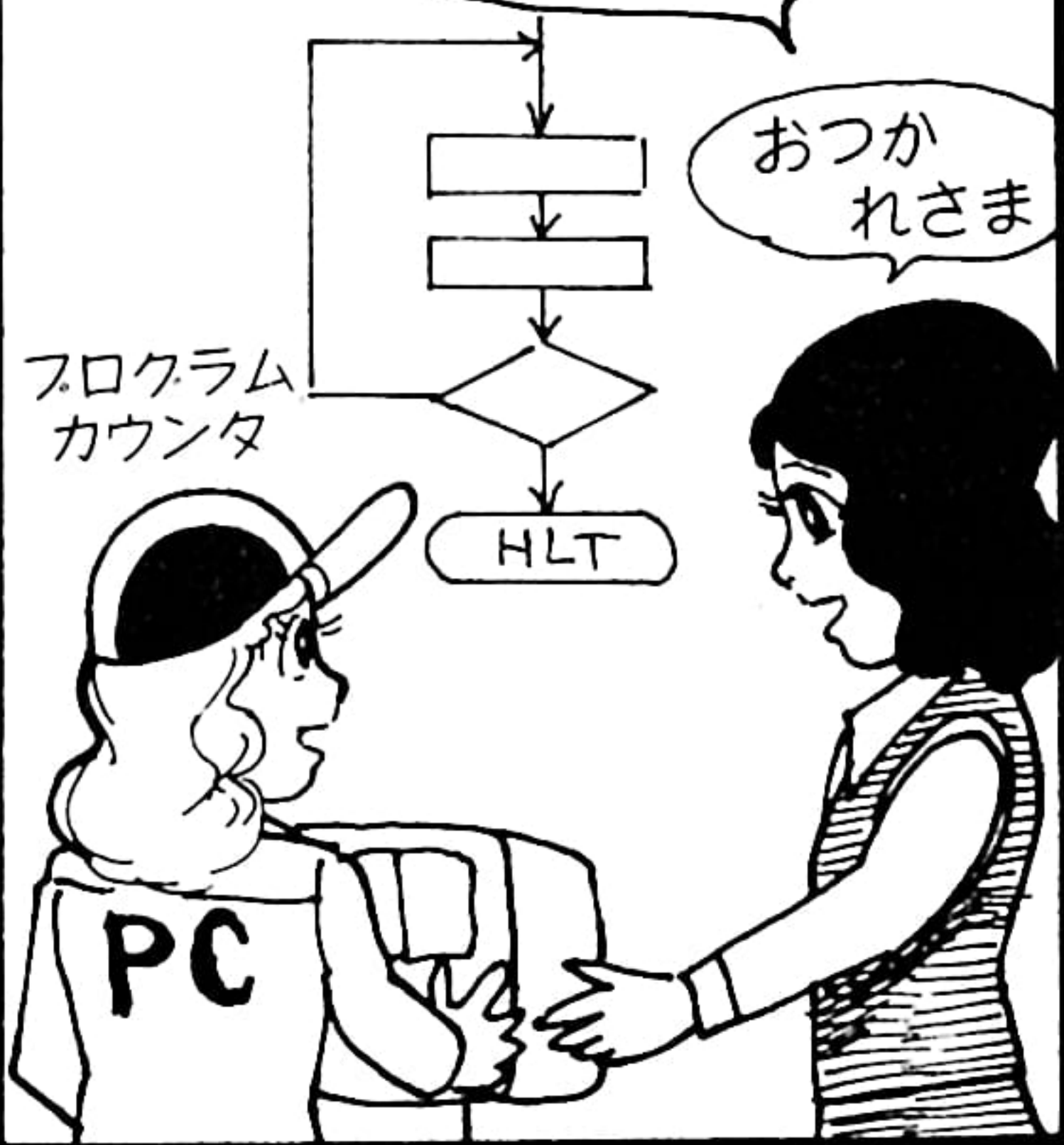
データ(A)
06
デバイスアドレス
0
1
2
FF
アウトプットポート
CPU
OUT
ステータス
IN
0
1
2
FF
インプットポート
MW
メモリアドレス
0
1
2
3
FFFF
メモリ

インクリメント、デクレメントという命令は本当にコンピュータ的なんです。使いこなせばこんな便利な命令はありません。

DCR C
ZERO
JNZ
ならば ZERO

IN␣01
DB 01
デコード
INP
INPUT PORT No.01
アドレスバスの上位8bit 下位8bitにこの数値が出力される
00010011
13
D0 1
D1 1
D2 0
D3 0
D4 1
D5 0
D6 0
D7 0
くわしいことは、本文を読んでくれよな。

OUT␣01
D3 01
デコード
OUT
ラッチ No.01
OUTPUT PORT
D0 0
D1 0
D2 0
D3 1
D4 1
D5 0
D6 1
D7 0
58

HLT (ホルト) 76

何事も始めがあれば終わりがあります。この命令の実行後はリセット信号をハード的に入れるまでプログラムの実行は停止します。

プログラムカウンタ
HLT
おつかれさま
PC

NOP (ノーオペレーション)

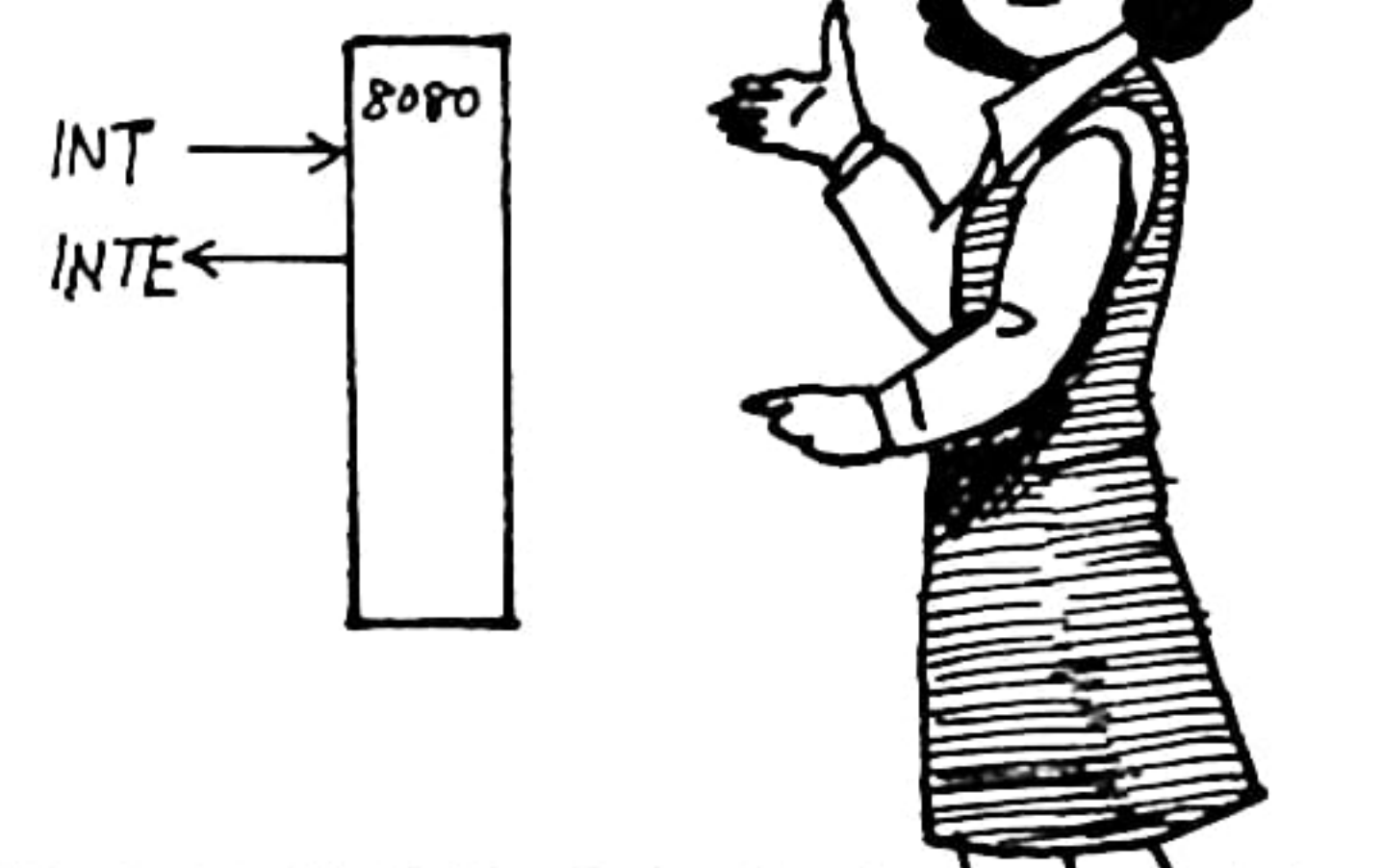
00
NOP
NOP
NOP
NOP
プログラムは進んでいくのです

この命令はなにもしないという変な命令です。

でも何もしないという命令を読み込むために4ステート分の時間を消費しました。これをあれこれ応用するんです。

EI
FB
(インタラプトイネーブル)

この命令を実行させておかないと外部からの割り込みができません。

INT
INTE
8080

DAA命令を使って16進数を10進数に変換する。

BCD下桁

BCD上桁

(A)

1101	1011

A

BCDコードではA、B、C、D、E、Fは禁止コードです。

0～9以外のコードだったら6を加えて0～9にします。

16進	BCD
(A) 1010	→0000 (0)
(B) 1011	→0001 (1)
(C) 1100	→0010 (2)
(D) 1101	→0011 (3)
(E) 1110	→0100 (4)
(F) 1111	→0101 (5)

CARRY

桁上げ分を上桁に加える

ALUの演算は8ビットの加減算のみで、答は16進数です。10進数の答がほしい時は、このDAA命令を計算命令の直後に入れておけばOKです。

DAA
27

ALUはCPUの中にあります。

(A)←(Ā) これはどんなことに使うのかというと、減算を加算で行なう場合です。この命令と、インクリメント、デクレメント、それから加算を組み合わせると減算ができるのです。CMA

5
-6 →
-1

↓ Aレジスタに入れる
0110
補数をとる
↓
1001
↓ 2の補数にするため
インクリメント(+1)
1010
加算する
↓
0101 (5)
+) 1010
キャリー → [ ] 1111
桁上げがなければ答は負数である
↓ 負の場合は答をデクレメント(-1)
1110
↓ 補数をとる
0001
∴答は－1である。

```
      1001  (9) …①
x)    0111  (7) …②
      1001
     10010 ……… ①を左に1ビット回転
+)  100100 ……… ②を左に2ビット回転
    111111
    3    F
```

16進数なので10進補正のDAA命令を実行すると、

3×16＝48
15×1＝15
63

63になります。

9×7＝63ですね。

乗・除算は左右の回転命令と加算命令の組み合わせでできるんです。

基本的にはそうですが、実際のプログラムは長くなるので省略します。

回転命令？ この変な命令にもちゃんと用途があるんです。乗除算のためになくてはならないものです。

RLC
07

FLAGS

	1
	CY

0	0	1	0	0	1	1	0
D7	D6	D5	D4	D3	D2	D1	D0

命令実行後 ⇩

	0

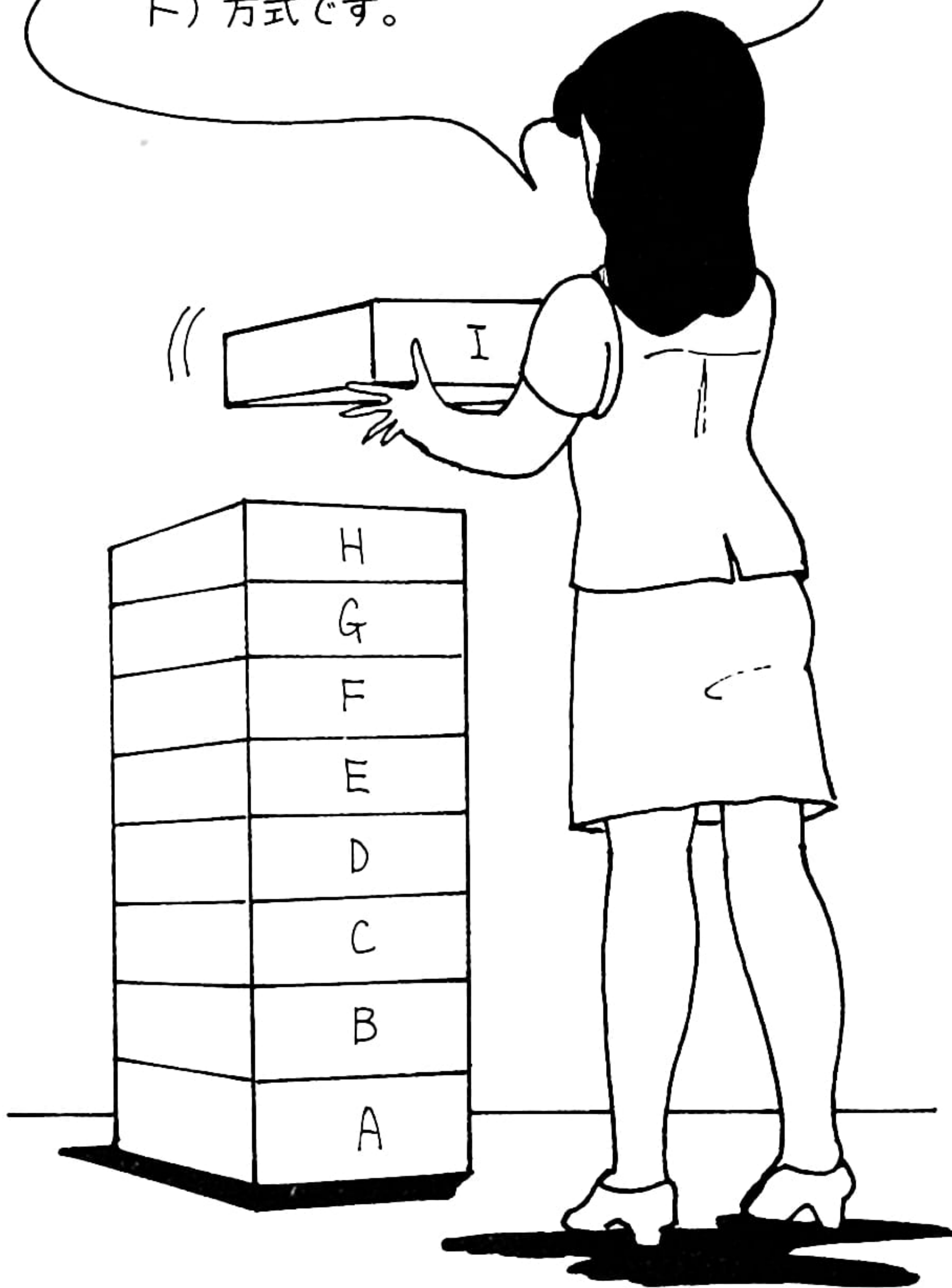
0	1	0	0	1	1	0	0

(C)←D7　　1ビットずらす　(D0)←(D7)

無条件ジャンプ、条件付ジャンプ、CALL、RETURNにはプログラムカウンタ（PC）ちゃんとスタックポインタ（SP）ちゃんがが対応します。
CALL
RET
J
HLT
スタックポインタ
プログラムカウンタ
PC
SP
JMPというのは無条件つまりフラグの状態などとは関係なしにジャンプすることです。RST命令によるベクトル・アドレスから割込みプログラムへジャンプさせる場合などに使います。
JMP
RST3
RST4
ベクトルアドレス
Z＝1ということは、ゼロフラグが立っているということです。この時ジャンプするのがJZ命令です。ジャンプ命令で最も多く使われるのが、このJZそれからJNZ命令でしょう。
"1"
ZERO
JZ <B2><B3>
ならば <B3><B2> 番地へジャンプ
JZ
DCR
HLT
JMP
ジャンプする前に、次のOPコードのアドレスをRAMに書き込んでおくのです。ジャンプ先のプログラム実行中にRET命令が出てくるとジャンプ前にメモったアドレスに再びジャンプ（戻る）するのです。
RAMちゃん
RAM
<B3><B2>
私はちょっとの間2つ若くなるの。
スタックポインタ
プログラムカウンタ
2014
メモリー
コール CALL
2011 op
2012 D7
2013 22
2014 op
<B3><B2> へジャンプ
22D7 op
22D8
サブルーチン
リターン RET op
SP こ
プッシュダウン
CALL命令実行後のプログラムカウンタの内容
CALL ␣ 22D7
CD D7 22
ジャンプ命令みたいだけどよく見るとちょっと違うのね。
次のオペコードのアドレスを 22D7 とする
戻り番地をメモる
2014

最初のスタックポイントはソフトで指定しておかねばなりません。
下位アドレス（0000）
スタック方向
アドレス
1バイトのデータ
上位アドレス（FFFF）
RAM
SP
RAMちゃんよろしくね。
CALL命令とRET命令は必ず対になっていなければなりません。
CALL
RET
プッシュダウンスタックとはPUSH・CALL命令でスタックポインタの示しているRAM領域のことです。PUSH命令を実行するとSPの内容は2つだけダウンするんです。
プログラム
LXI SP 20CD
PUSH B
PUSH D
PUSH H
PUSH PSW
(SP)
down
スタックポインタの設定
フラグ 20C5
(A) 20C6
(L) 20C7
(H) 20C8
(E) 20C9
(D) 20CA
(C) 20CB
(B) 20CC
20CD
20C4
RAMメモリ
しかり、私(POP)はすぐに来る。
PUSH PSW
(A)
FLAGS
RET
CALL
PUSH B
(B)(C)
PUSHとは退避せよってことかな？
PUSH命令は4種類あるの
(A)
(FLAGS)
POPよきてください
(B)(C)
私たちは死んでも再び生きます。アーメン。
PUSH D
(D)(E)
いざ。
アドレス
down
(D)(E)
では、戻れという命令もあるはず？
(SP)
PUSH H
(H)(L)
(H)(L)
プッシュダウン スタックと呼ばれるRAMスタックエリア（長さは不定）
RAM

こういう積み方だと後に置いたものから先に取り出さねばならないですね。これがLIFO（ラストインファーストアウト）方式です。

I
H
G
F
E
D
C
B
A

PUSH命令で退避させたデータを元に戻すのがPOP命令です。後から入れたものを先に出す(LAST IN FIRST OUT)しくみになっています。

プログラム
①PUSH B
②PUSH D
③PUSH H

(H) (L)
B A D C
(D) (E)
6 7 9 8
(B) (C)
2 1 4 3
SP
DC
BA
98
67
43
21

この場合も、プッシュダウンスタックに、戻る番地を入れておき、割込みプログラムの終わりに書かれたRET命令によってメインに帰ります。

RST命令は、ハード的に割込みポートから挿入される1バイトのベクトルアドレスジャンプ命令です。

RSTn
Rst
RET
RET

ああ忙しい

この場合は、HLレジスタペアからデータを戻します。順番を間違えると、データも別のものが入ることになります。

PUSHとPOPは対にして使われるのです。

プログラム
④ POP H
⑤ POP B
⑥ POP D

順番を間違えたか、またはわざとデータチェンジのために入れ換えたのかどちらか。

(H) (L)
B A D C
(B) (C)
6 7 9 8
(D) (E)
2 1 4 3
DC
BA
98
67
43
21
SP

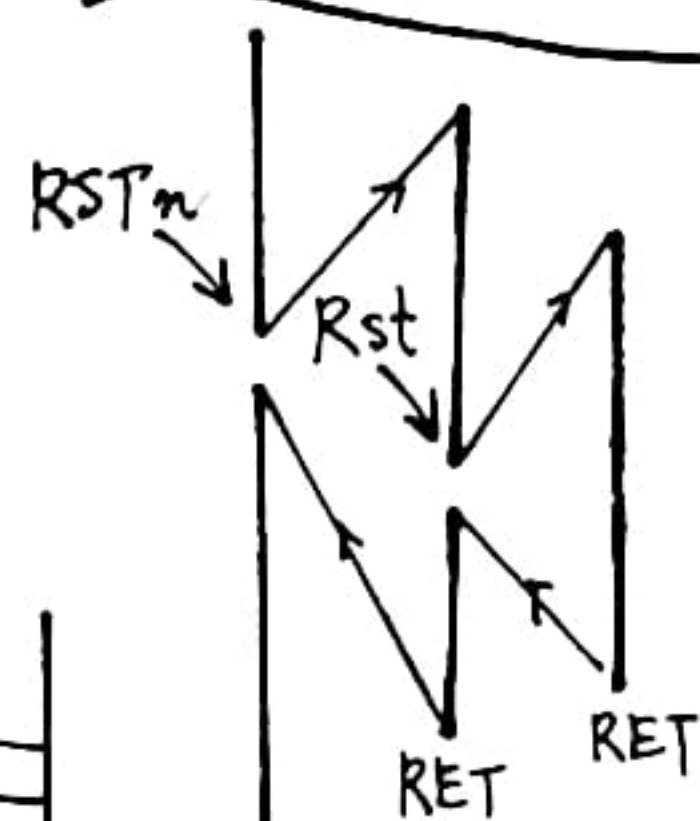
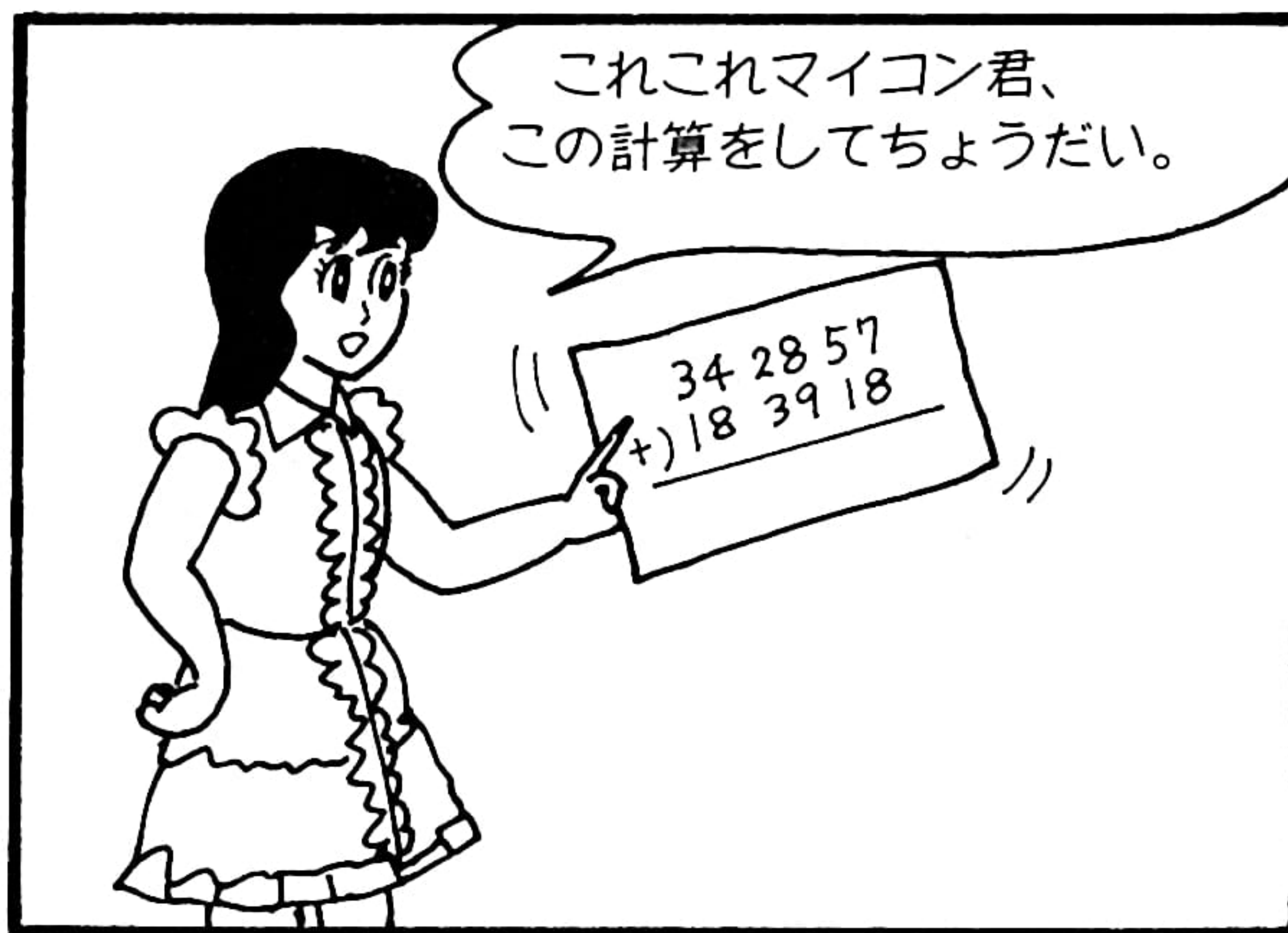
⑧マシン語でプログラムだ！

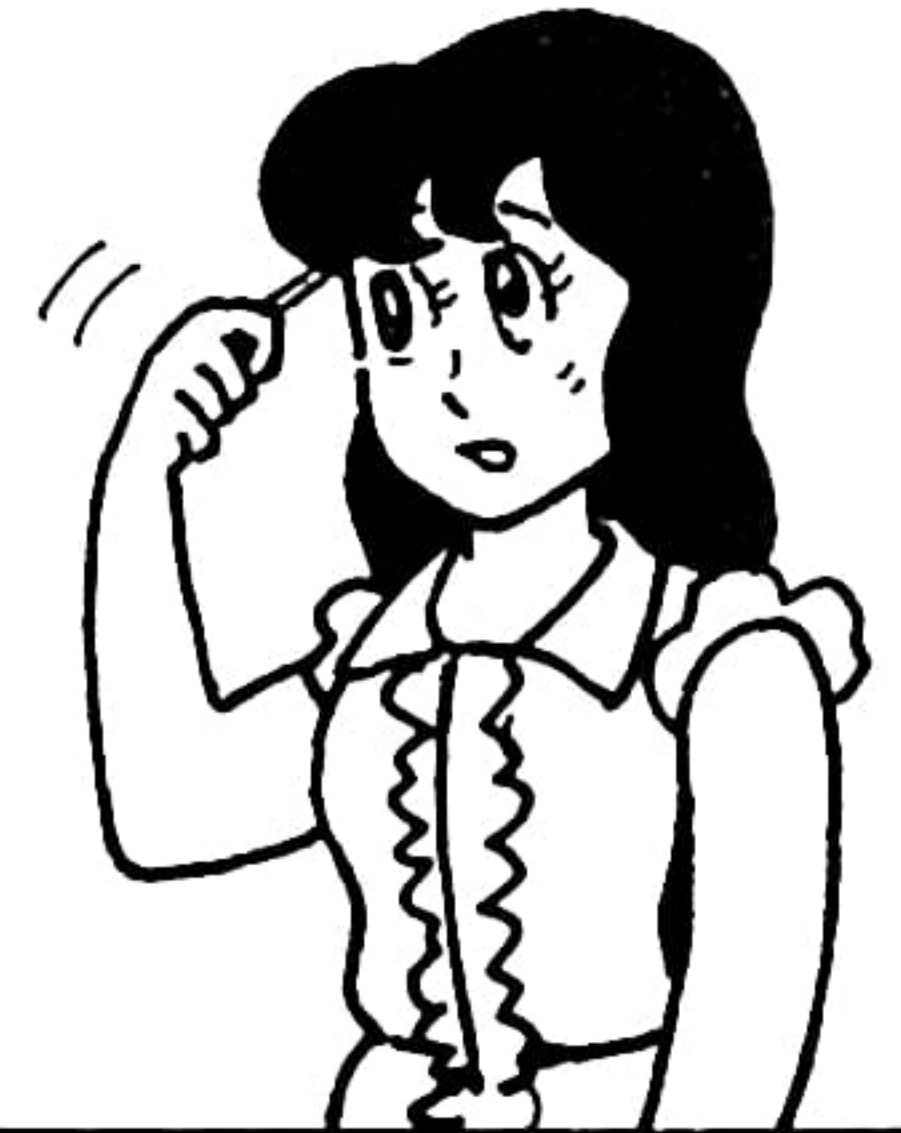
342857
+ 183918

人間がやる方がずっと簡単だけど…

これこれマイコン君、この計算をしてちょうだい。

342857
+) 183918

と言っても動くわけないわね。マイコンにわかるマシン語でのプログラムにチャレンジするしかないようね。

プログラムの内容は被加算数3バイトを2000番地からメモリ、

アドレス	
2000	57
2001	28
2002	34

34 28 57

1メモリは1バイトに対応

続いて加算数3バイトを2003番地からメモります。

2000	57
2001	28
2002	34
2003	18
2004	39
2005	18

18 39 18

次に下桁から1バイトずつたし算し、10進補正して結果を被加算数を入れておいたメモリに戻します。

2000 57
2003 18

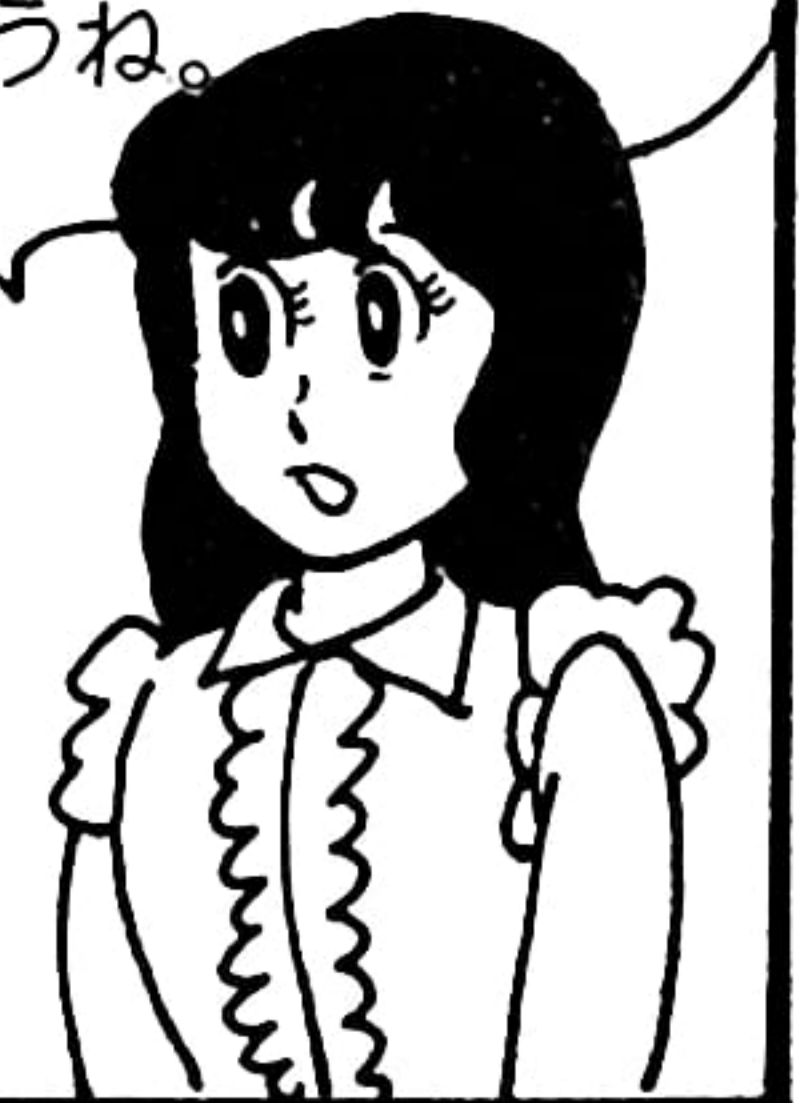
57
+18
75

答を戻す

答は被加算数の3バイトと置きかわって2000番地からの3バイトに入っていることになります。

2000	75
2001	67
2002	52

答

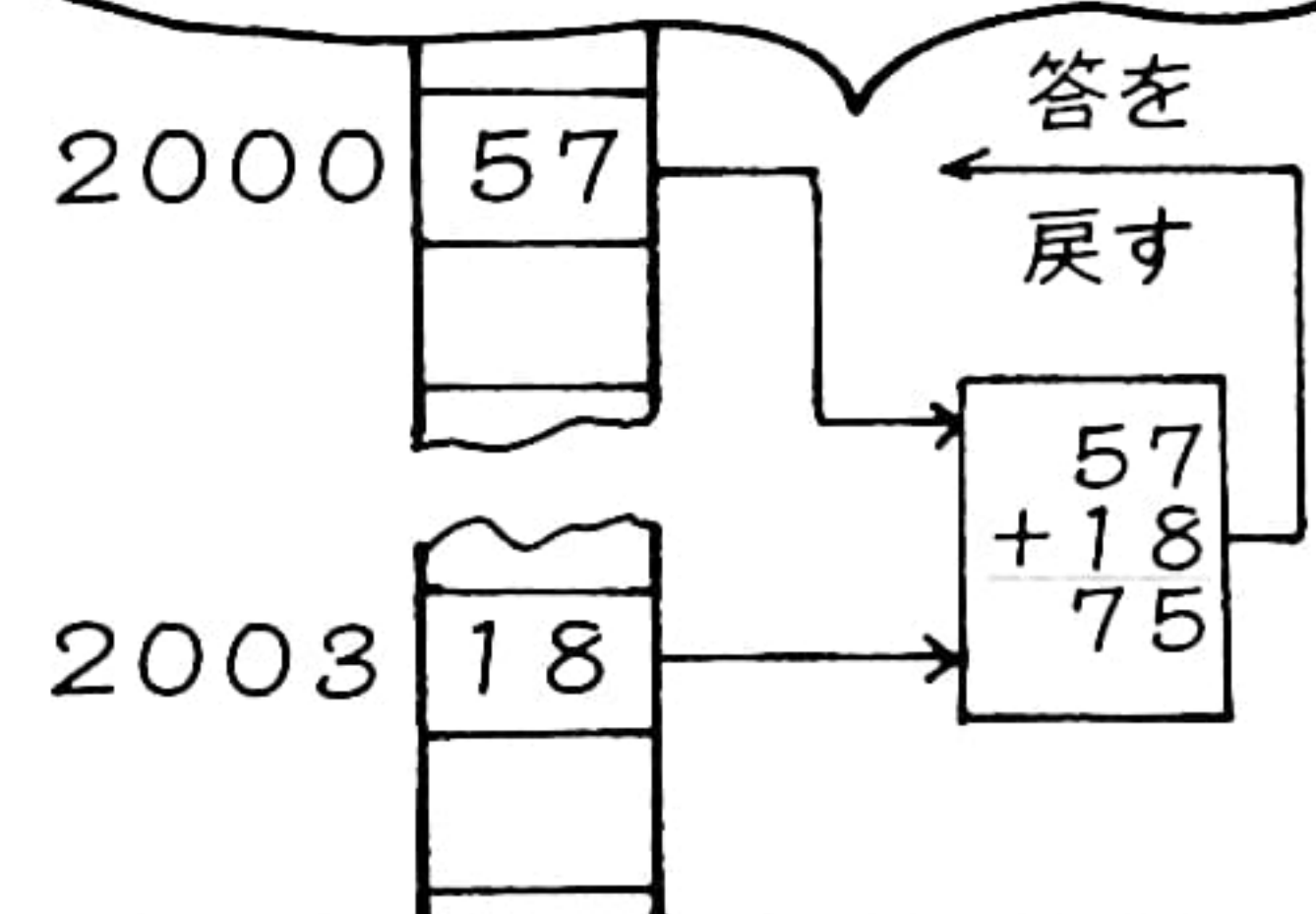
57
28
34

ここではLEDや画面に答を表示させるところまではやりませんでした。OUTPTサブルーチンを使えばできるはずです。

526776

プログラムはまずニーモニックでコーディング（記述）し、それからマシン語に変換します。

```
ORG  2011
LXI  B,2000
LXI  H,2003
XRA  A
MVI  E,03
 ⋮
```

人手でニーモニック → マシン語に変換することをハンドアセンブルといいます。

LXI␣B,2000→210320
LXI␣H,2003→

ニーモニック
オペコード
変換が必要

ハンドアセンブルの場合、コーディングの形式なんてどうでもいい気がしますが正式にやっておいた方があとでアセンブラを使う時、二度手間にならずに済みます。

スペース　コメント

LOOP：LDAX␣B;

ラベル　ニーモニック　オペランド

ラベルはジャンプ先のアドレスを記号で表わすやり方でアセンブリ言語のコーディングがしやすくなります。

LOOP：LDAX␣B

JUMP␣LOOP

マシン語のアドレスがわからなくてもコーディングできる。

プログラム例

アセンブリ言語

	ラベル	ニーモニック	オペランド	コメント
		ORG	2011	;最初のオペコード
①		LXI	B, 2000	;被加算数のアドレス
②		LXI	H, 2003	;加算数のアドレス
③		XRA	A	;キャリーのクリア
④		MVI	E, 03	;ループ回数のセット
⑤	LOOP：	LDAX	B	;Aレジスタにロード
⑥		ADC	M	;加算
⑦		DAA		;10進補正
⑧		STAX	B	;Aレジスタの答を移す
⑨		DCR	E	;(E)=(E)−1
⑩		JZ	DONE	;(E)=0でDONEへ
⑪		INX	B	;(BC)=(BC)+1
⑫		INX	H	;(HL)=(HL)+1
⑬		JMP	LOOP	;上桁計算へ
⑭	DONE：	HLT		

では1行ずつハンドアセンブルしていきましょう。2011番地に最初のオペコード01が入ります。

このプログラムを走らせる前にデータをキーインしておかなくてはなりません。答もキー操作でアドレスのデータで確認します。

アドレス データ
2000−57 2003−18
2001−28 2004−39
2002−34 2005−18

␣はスペースを示します。

まず被加算数の下位番地をBCレジスタに入れます。これは3バイト命令です。

LXI␣B, 20 00
01 00 20

番地 データ
2011 010020

加算数の下位番地はHLレジスタに入れます。

LXI␣H, 20 03
21 03 20

マシン語です。

番地 データ
2014 210320

計算回数をセットします。

MVI␣E, 03
1E 03

番地 データ
2018 1E 03

3回目 2回目 1回目
34 28 57
+ 18+39+18

これは1バイト命令です。

キャリーの補助フラグと桁上げフラグをリセットします。

XRA␣A
A F

番地 データ
2017 AF

メモリから僕、すなわちAレジスタへのデータ転送命令です。これは1バイト命令です。
(A)
57
[(B)(C)]
57
データを転送する
RAM
A
2000
B
C
2000
番地を指定する
LOOP：LDAX␣B
0A
ラベルです。
アセンブリ言語による番地の指定です。あとで必要になります。
番地　データ
201A　0A
⑤

いよいよ計算開始です。
57　0101 0111
+18　+) 0001 1000
57
私はALUである。1バイトの加算をします。
ADC␣M
8E
"0"
(A)←(A)+(M)+(CARRY)
ALU
RAM
(M)
[(H)(L)]
18
2003
2003
⑥
番地　データ
201B　8E

キャリーはありません。
CY
"0"
CPU
AC
"0"
つまんないの。

57＋18＝75じゃないかと思うだろ？　私は16進数の計算しかできないのです。
計算結果はAレジスタに戻します。前のデータ57が6Fに変わります。
0101 0111
6F　+) 0001 1000
0110 1111
(6)　(F)
A
キャリーなし
キャリーなし
補助キャリーなし
ALU

まず6Fつまり01101111の下位4ビットを見て、これが10以下か補助フラグが1だったらこのFに6を加えます。
DAA
27
⑦
番地　データ
201C　27
1111 (F)
+ 0110 (6)
1 0101 (5)
キャリー
AC
キャリー　6を加えた結果
補助キャリーが"1"になる。

ここで16進 → 10進補正命令を使います。

結果は再びAレジスタに入れます。
さらに上位4ビットを見るとこれは6で10以下だしキャリーも0ですから補正のための6は加えません。ただし、さきほどACフラグが1になっているのでこれを加えて7にします。これでDAA命令の動作は完了です。
75となる。
ウェーン えこひいきするう〜。
シーッ
こら
4ビット目
0110
+ 1
0111 (7)
75が入る
6F
A
ALU
CY
AC

桁上げが起こるので私がフラグを立てます。
その結果答は5になり、
"1"
サッ
"0"
AC

Dさん今ごろどうしているかしら？
ここまでの過程でレジスタとメモリの内容はこう変化しています。
シンドくなってきた…
03
E
H L
20 03
75
A
B C
20 00
18
57
2003
2000
AC
CY

これで下2桁の計算ができました。
57
+ 18
75

いいすよ。減るもんじゃないし。
このデータはそのまま残る
データをちょうだい。
57は消える
STAX B
75
75
A
2000
20 00
02
番地 データ
201D 07
⑧
02を07としてるのは単なる転記ミス

Aレジスタの答はメモリの2000番地に入れることにします。

デクリメントは1だけ減らす変な命令です。
(E)←(E)-1
DCR E
あっ、ドロボー！
3-1
02
E 011
命令なんだ悪く思うなよ。
いただき
1D
⑨
番地 データ
201E 1D

同じことをあと2回やればいいのですが、ここでループの手法を使ってコンピュータ的にプログラムしてみます。

私はジャンプ司令官のひとりでゼロフラグの状態を見てジャンプするかどうか決めます。ゼロフラグが旗を上げたら2バイト目、3バイト目の示すアドレスにジャンプさせます。

JZ␣DONE

CA □□ □□

番地　データ

201F　CA □□ □□

JZ

ゼロ

私はゼロフラグです。実行された演算の結果を見ています。今の計算はE＝3－1＝2ですからゼロではないので旗は上げません。

ゼロ

これはラベルといい番地専用の変数みたいなものです。この番地は今はわかりません。

ここではジャンプ先が数値でなくDONEという文字で示されています。

DONE

プログラムカウンタ

ゼロフラグが旗を上げない時は2バイト先のアドレスが次のオペコードとなります。

なかなか大変なのです。

この場合DONEのラベルを持つ文は先にあるのでここは残しておかねばなりません。

ADC M　201B　8E

DAA　201C　27

STAX B　201D　07

DCR E　201E　1D

JZ　DONE　201F　CA □□ □□

ジャンプ先の番地が入ります。

"1"

E=E-1

E=0?

DONE

YES

NO

INX B

"0"

INX H

このJZは条件付きジャンプです。フローがここで分岐します。

(B)(C)←(B)(C)+1

INX␣B

03

女はよくばりだなあ

たった1だけ？

おうけとりください。

番地　データ

2020　03

⑪

+1

B

2 0 0 1

1

レストラン8080

では、ハンドアセンブルを続けましょう。

ここがジャンプ先です。

INX␣B

INX␣H

JMP␣LOOP

DONE HLT

よろしく。私は無条件ジャンプ司令官です。このループも番地を示すラベルです。
JMP␣LOOP
C3 □□ □□
B2 B3
Loop
(B3) (B2)
2 0 1 A
番地 データ ⑬
2024 C3 1A20

(H)(L)←(H)(L)+1 INX␣H
23
＋1はインクリメントというんだね。
ごくろうさま。
あの、これ。
番地 データ
2023 23
⑫
+
+1
20 01
(H) (L)

マシン語に直すとこの201Aは1A20となります。
JMP␣201A
C3 1A 20
この理由はCPUが下位アドレスを先に読むからです。
C3
1A
20
下位アドレス

アセンブラ語		機械語	
ラベル	ニーモニック	番地	データ
	XRA␣A	2017	AF
	MVI␣E 03	2018	1E 03
LOOP :	LDAX␣B	201A	0A
	ADC␣M	201B	8E
	DAA	201C	27
	⋮		
	JMP␣LOOP	2024	C3 1A 20

このLOOPはすでにアセンブルした所にあります。この場合LOOPは201A番地です。

ここでさきほどのDONEの番地が決まりました。2027番地です。

ラベル ニーモニック
DONE␣HLT
76
ねえどうしたの？
ウーン。何もしてはいけないんだって。
NOPではない
番地 データ
2027 76
⑭

これでハンドアセンブルが完了しました。
JZ␣DONE 201F CA 27 20

ハッ
きみ、これを
ごくろうでした。
JZ司令官に報告します。
DONE
2027
JZ

マシン語		翻訳	アセンブリ言語		
番地	データ	ハンドアセンブル	ラベル	ニーモニック	オペランド
		⇦		ORG	2011
2011	01 00 20			LXI	B,2000
2014	21 03 20			LXI	H,2003
2017	AF			XRA	A
2018	1E 03			MVI	E,03
201A	0A		LOOP:	LDAX	B
201B	8E			ADC	M
201C	27			DAA	
201D	07			STAX	B
201E	1D			DCR	E
201F	CA 27 20			JZ	DONE
2022	03			INX	B
2023	23			INX	H
2024	C3 1A 20			JMP	LOOP
2027	76		DONE:	HLT	
2000	57 28 34			DB	57 28 34
2003	18 39 18			DB	18 39 18

ホントは間違っている

このマシン語プログラムはモニタつきのパソコンでもアドレスを変えれば走ります。ただしCPUが8080かZ80の場合です。

PC-8001ならS、G、Dのコマンドを使います。

MON
*
SC00

あなたも8080、Z80マシンでやってみてください。

このプログラムをパソコンでアセンブルするのはあとです。オールRAM型パソコンで、CP/Mプログラムの走るものを使います。ここではSDK-85を使いアドレスとデータを1つづつ示していきます。

おわかりと思いますがG(ゴー)させる前にはC000番地から57、28、34、18、39、18のデータを入れておかねばなりません。

PC-8001ならORG␣C011としてもう一度ハンドアセンブルしてください。FM7/8などはアドレスだけでなくマシン語も変える必要があります。

まず、

RESET SUBST MEM 2 0 0 0 NEXT とキーインしました。

2000番地が表示されます。

アドレスを表示する　データを表示する

2000 E5

この表示は、前のものでキーインして57にします。

次に 5 7 と押すと2000番地に

57が入ります。

2000 57

2000番地に57を入れようとしている。

これで計算用のデータをメモリの中に入れることができました。

34 28 57
18 39 18

続いて 2 8 NEXT 3 4 NEXT 1 8 NEXT 3 9 NEXT 1 8 NEXT と押していきます。

2006 08

アドレスはインクリメントされて2001が表示されます。

NEXT を押すと

2001 38

モニタプログラムにより自動的に＋1されている。

意味のない前のデータ

うまくいけば、2000番地から3バイトに答が入っているはずなんです。ではプログラムをRUNしてみましょう。

2000 75
2001 67
2002 52

… 2 0 NEXT 7 6 NEXT で終わりです。なかなかしんどいですね。

2028 78

次はプログラムを入れます。RESET SUBST MEM 2 0 1 1 NEXT 0 1 NEXT 0 0 NEXT …と順に入れていきます。

RESET SUBST MEM 2 0 0 0 NEXT で答えを確かめます。

元のまま

2000 57

このEはエラーじゃなかったんだっけ

ああ、一瞬ドキリとしたりして……。

?

RESET GO 2 0 1 1 EXEC でプログラムで走るはずです。

E

いつ私が足を組んだりしたのっ！

いけね、またやっちゃった。

おっとっと

作者

う～ん、どこがいけないんだろう？

ガクッ、やっぱりだめか…。

期待した通りにプログラムが走らなかった時、その原因を調べて誤りがあれば修正しなければなりません。

デバッグ作業
(虫とり)

原因その1

プログラムそのものに問題がある時。

START

原因その2

ハンドアセンブルの際の変換ミス。

STAX B　番地　データ
201D　07
02

こんな信じられないようなミスをすることもあります。

原因その3

キーインする時のミス、などがあると思います。

もう一度リストを見ると、ありましたね。
印の07は実は02でした。

程度の差こそあれ…。

プログラム全体を通してこんなミスがあるものです。これを虫といい、ひとつづつつぶしていかねばなりません。

その他のミスはなさそうですから201D番地の内容を変更します。

RESET SUBST MEM 2 0 1 D NEXT 0 2 NEXT で修正完了。

この表示は、このキーを押した時のものです。

201d 07

これはDです。

さてもう一度アタック。RESET GO 2 0 1 1 EXEC と押す…。

するとただちにEとなる。

E

RESET SUBST MEM 2 0 0 0 NEXT むむ、75だ!

いいぞ。

34 28 57
+18 39 18
75

2000 75

NEXT

67!

34 28 57
+18 39 18
67 75

2001 67

+1ずつ増えていくのをインクリメントといいます。

NEXT

52! やった!

34 28 57
+18 39 18
52 67 75

2002 52

こんどはバッチリでした。

ところで。

この計算は何秒かかったのでしょう。

ステート数を計算すればおよその時間がわかるのです。今SDK-85の１ステート分の時間がわかりませんのでNECのTK-80の時間でやってみます。

キットTK-80の1ステートは、0.4882812 μSEC

ステート数は命令によって決まっています。

プログラムのメモリの容量と、実行時間とは関係ないのでご注意ください。

ここでの回数はループのルーチンで何回も同じ命令が実行されたことを示すものです。ステート数の合計は各命令の実行回数を考えねばなりません。

フワァねむいニャ

ニーモニック	ステート	回数	計
LXI B	10	1	10
LXI H	10	1	10
XRA	4	1	4
MVI E	7	1	7
LDAX B	7	3	21
ADC M	7	3	21
DAA	4	3	12
STAX B	7	3	21
DCR E	5	3	15
JZ	10	4	40
INX B	5	3	15
INX H	5	3	15
JMP	10	3	30
HLT	7	1	7

合計 288ステート × 0.4882812 = 140、62498 μSEC

このプログラムは288ステート必要で実行時間は約140 μSECです。すごいスピードですね。

ピンとこないけど…。

1 μSEC = $\frac{1}{1000000}$ SEC マイクロ秒

このスピードと信頼性がとても魅力的です。

0.00014 SEC

342857
+ 183918

おれなら5秒はかかるナ…。

チッおまえに

そうじ、せんたく、料理ができるかよ。

(この時分作者はチョンガーでした)

それにもう１つの大きな魅力は判断機能があることです。

この判断機能があるため昔は電子頭脳なんて言われたわけです。そのメカニズムをもう少し説明しましょう。

? YES NO

判断機能は、フラグ操作命令とその次に置かれる条件付きジャンプ命令との組合せでできます。

プログラムの流れがよくわかりますね。
START
2011番地からプログラム
① LXI B, 2000
② LXI H, 2003
③ XRA A キャリーをリセット
④ MVI E, 03 E=3とする
⑤ LDAX B (A)←(B)(C)
⑥ ADC M (A)←(A)+(M)+CY
⑦ DAA 10進補正
⑧ STAX B [(B)(C)]←(A)
⑨ DCR E E=E−1
⑩ E=0 YES (CYフラグ"1")
(CYフラグ"0") NO
⑪ INX B
⑫ INX H
⑬
⑭ HLT
STOP
このようになります。これをフローチャートと言っています。
これはフローチャートを書く時に使われる判断記号です。
□記号は処理記号です。今のプログラムを□と◇で表現してみましょう。
1
0
サイン
ゼロ
パリティ
キャリー
演算の結果、判断できるフラグはサイン、ゼロ、パリティ、キャリーの4つです。
ミシッ
サインジャンプ
ゼロジャンプ
無条件ジャンプ
パリティジャンプ
キャリージャンプ
このプログラムはゼロフラグを参照してジャンプするJZを使いましたが…。
DCR E
E=0
YES
番地DONEへジャンプ
NO
INX B
そのほかにもサイン、パリティ、キャリーを参照するジャンプ命令もありますし、JZとは逆のJNZというようなものもあります。

NANDゲートを用いたフリップフロップの例
"1"
Vcc
SET
OUT
RESET
SETが押されると、次にRESETが押されるまで白は"1"を保ちます。
白と黒は、ギッタンバッタンするわけです。
"0"
"0"
"1"
フリップフロップはシーソーのギッタンバッタンということです。本当にそういう感じです。
フラグというのはフリップフロップのことです。
ゼロ
CPU

このうち一番よく使われるのは何と言ってもゼロフラグジャンプ命令でしょう。

プログラムをつくる時に都合のいいのを使えばいいわけです。
ジャンプ条件はフラグが'1'のときでも'0'のときでもいいように2つの命令があります。
1でジャンプがJZ命令
0でジャンプがJNZ命令
ゼロ

ゼロフラグは演算の結果が0になる時
'1'の信号を出します。
1
0

そして、自分の条件が満たされると、プログラムカウンタをいじって、
2バイト目
3バイト目
1バイト目はOPコード
CA
1
0

この考え方はループ処理、BASICならFOR～NEXT文というところです。
ループ

こういう考え方になじんでくるとあなたの頭もかなりコンピュータ的になってきたわけです。

フラグ操作と条件付ジャンプ命令をうまく組み合わせて行ないます。
判断は…。
フラグ操作を伴う命令
フラグを見てジャンプ
この命令の実行後フラグは1か0かのどちらかですから、プログラムに都合のよい条件付き（フラグを見ること）ジャンプ命令を入れておきます。これがコンピュータの判断機能です。

ループから抜け出すのです。
プログラムカウンタ
3バイト目
2バイト目

⑨プログラムカウンタとスタックポインタ

よく使うサブルーチンというプログラムはCALLとRET命令をセットで使います。これからこの命令実行時の裏方さんのお話をします。

SUBROUTINE

A	F/F
B	C
D	E
H	L
PC	
SP	

メモリの番地	メモリの内容データ	ニーモニック
⋮	⋮	
201A	CD B7 02	CALL␣OUT PUT
⋮	⋮	
202A	CD F1 05	CALL␣DELAY
⋮	⋮	

サブルーチンを呼ぶというのはソフト屋さん的な言い方です。

'HELP' プログラムでも 'OUTPT' と 'DELAY' の2つのサブルーチンを使いました。

でも、ただのジャンプじゃないんですよね、これが。

'HELP'という名前のプログラム

メインプログラム

'OUTPUT'という名前のサブルーチン

201A CD
201B B7
201C 02
201D 3E
201E 00

02B7 0F
02B8 DA
⋮
02E5 02
02E6 C9

OUTPTはラベルでそのアドレスは02B7番地です。CALL文が実行されるとここにジャンプするわけです。

よろしく、私はプログラムカウンタです。

CPU

このことを理解するためにはプログラムカウンタとスタックポインタの働きを知る必要があります。

それは、サブルーチンの処理が終わったあと、ジャンプした命令の次の命令に戻ってこなければならないということです。

リターン
RETURN

割込み8レベルのベクトル・アドレスの都合で、RESETで0000にした時8バイトしか使えないのでジャンプ命令で0040(H)以降に飛ばします。
op
RAM
0000

ふわーい。電源リセットで私の内容は0000となり、プログラムは、ここからスタートです。RESETは、一番強力な割込み入力なのです。
リセット
ON
0000

私は16ビットのカウンタなのです。ふつうはマシンサイクルのたびに1カウントするのですが、命令の種類によっては、しない時もあります。
2038

そして、ジャンプするというのはこのプログラムの内容を何かの手段で変更してやることじゃないかしら？

そうか！　キットで表示されていたアドレスというのはこのプログラムカウンタの内容だったのね。よくわかったわ。
2038 75

0000
ジャンプ
ベクトル・アドレス部
割込み命令RSTで使用する
0040
メインプログラム

データバスへ

W (8)	Z (8)
B (8)	C (8)
D (8)	E (8)
H (8)	L (8)
PC (16)	
SP (16)	
インクリメンタ・デクリメンタ アドレスラッチ	

指示
アドレス・バス

アドレス・バスに出力されるデータは、プログラムカウンタ(PC)の内容とは限りません。実行過程ではいろんなレジスタの内容が出力されます。
指示書
op

私は、インクリメンタの指令によってカウントするだけです。プログラムのデータを読み込むために私の内容をアドレス・バスに出力するのは別の人たちがやるのです。
アドレス・バス
指示書(PC)

命令で私たちがタッチできるレジスタではないんですけど、プログラムカウンタ（PC）やスタックポインタ(SP)を助けるレジスタがあります。ちょっと紹介しておきますね。

オペコードレジスタさん。RESET後の0000番地の内容を最初に読むのはこの人です。

ジャンプ、PUSH、POP、CALL、RET命令などでPCを助けます。W・Zレジスタちゃん。

アドレスラッチさん、この人のおかげでPCは安心してインクリメントできます。

アドレス・バス
W
Z
OP

オペコードは0番地からずーっとつながっているわけです。

またジャンプ司令官からのジャンプ命令やコール命令がくると、

OK

それまでのインクリメント作業を中断して指定されたオペコードの番地をセットするのも私の仕事です。

その後再びインクリメント作業に入るというわけです。これがプログラムジャンプの理屈です。

+1 +1

PC

プログラムの流れ

私はアドレス指定の専門家ですが、私のほかにもアドレス指定できるものがあります。オペコードフェッチ後の実行サイクルではBC、DE、HL、WZなど です。

B C D E H L W Z

RAM

XXX1 XXX2 XXX3

X X X 1

さらにもうひとりアドレスバスに自分のデータを出力できる仲間がいます。彼女の名前はスタックポインタ です。

やあ、はじめまして。

SP

0XロB

0Xロ7 0Xロ8 0Xロ9 0XロA 0XロB 0XロC

RAM メモリー

スタックとはほし草や棚を積み重ねるという意味なのね。私の作業はこれに似ているのよ。SPと略します。

最初のSPは、プログラムでLXI SP命令を使って指定してください。たとえば、最初 211A にSPを設定すると、

LXI SP, 211A

311A21

211A

この211A番地を基準として、CALL、PUSH、POP命令によってスタックポイトが移動します。

SP

211A

(2118)

(2119)

RAM

2119

SPが示している番地

211A

CDがオペコードとしてフェッチされるとまずPCちゃんが作業を始めます。

201A　CD B7 02
201D　3E 00

では、2人の紹介が終わったところでサブルーチンのしくみについて話してもらいましょう。

私の内容は必ず2ずつ減ったり、増えたりします。役目として　2バイトのアドレスを2バイトのメモリに退避させる必要があるからです。

+2
−2
SP 211A

一方、私スタックポインタは、SP←SP−1としてPCの上位1バイトをとり出します。

早くして手がしびれる

(W) (Z)
02B7

デクレメント
−1
2119
グサッ
20
201D

まずWとZの2つのレジスタにジャンプ先のアドレスを入れます（この2つのレジスタは外部からはさわれません)。

(CALL 02B7)
CD B7 02
(W) (Z)
02B7

プログラムカウンタは3バイト目の02を読むとこうなっている。

201D

スタッ
20
以前に入っていたデータは消えてしまう。
A
9
RAM
2119
(SP)
2119

これをまずSPが示している番地2119のメモリに送る。

PCの内容をRAMメモリに待避させる作業

エイッ
20
ピューッ
2119

その結果を、教科書的に表現すると、こうなります。

現在のスタックポインタの内容 2118

	番地	内容
	2117	XX
SP ⇨	2118	1D
ここ	2119	20
	211A	XX

0〜Fまでの何か

命令で最初にセットしたところ

これは、現在SPは2118番地となっているので2118番地に送ります。

A
1D
RAM
C
2118
(SP)
2118

さらに
SP←SP−1として今度はPCの下位1バイトをとり出します。

1D
1D

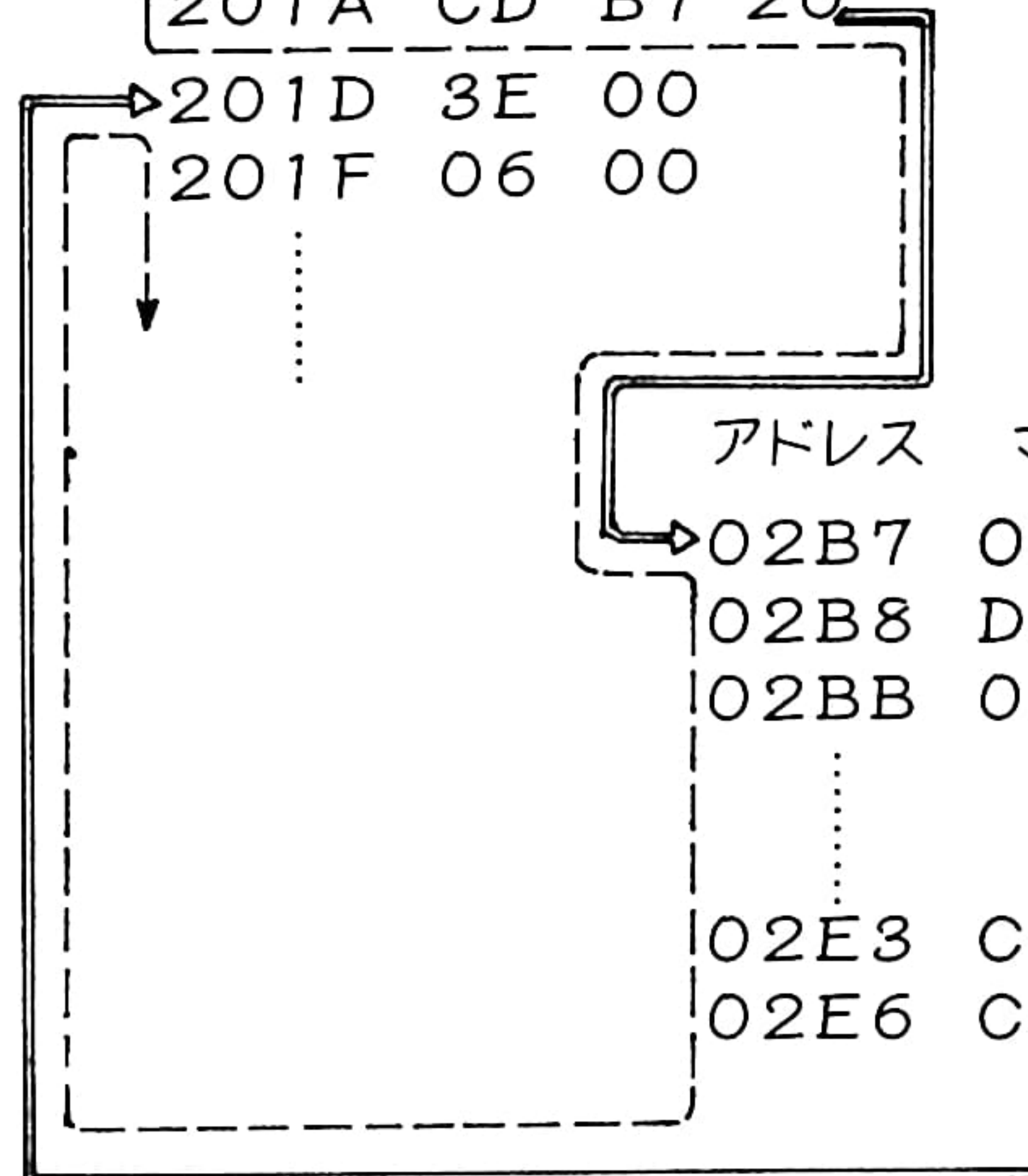
このあとは、PCを1ずつ加算、つまりインクリメント作業を続け、サブルーチンのプログラムを実行します。

+1

SDK-85では、02B7番地はROMメモリで、あらかじめモニタ・プログラムが入っています。その中のひとつのサブルーチン'OUTPUT'の最初のopコードが02B7番地に入っているわけです。

SPちゃんの作業が終わると私はW・Zレジスタに入れていたアドレスをPCカウンタに入れます。

'HELP'点滅プログラム(メイン)

```
          ニーモニック          アドレス    マシン語
     LXI␣SP,20C8H           2010    31  C8  20
     MVI␣A,1                2013    3E  01
     MVI␣B,0                2015    06  00
     LXI␣H,2004H            2017    21  04  20
     CAL␣OUTPUT             201A    CD  B7  20
DPY : MVI␣A,0               201D    3E  00
     MVI␣B,0                201F    06  00
       :                      :
```

ワンボードマイコンではこのOUTPUTの他にDELAY(時間調整)ルーチンなどをモニタの中に持っていてユーザーが使えるようになっています。

表示ルーチン(サブルーチン)

```
アドレス   マシン語          ニーモニック
02B7    0F              OUTPT : RRC
02B8    DA  C2  02              JC␣OUT05
02BB    0E  04                  MVI␣C,4
  :                               :
02E3    C2  C9  02              JNZ␣OUT15
02E6    C9                      RET
```

HELP

----→ はプログラムカウンタのマラソンコース

スタック ポインタ、リターン命令がきたわよ。

命令の解読(デコーダ)

はーい

そして、サブルーチン最後の命令はRET(リターン)です。この命令でメインプログラムに戻ります。その方法は?

SP
2118 1D 春子
2119 20 夏子
まず私(SP)の示している番地のメモリからデータをとりあげて
1D
SP
2118
RAM
春子
1D
2118
Zレジスタに入れます。
前のデータは追い出されて消える
ポイッ
メモカード
1D
0 2
W
Z
B
02E6
そして、
SP←SP+1つまり、
SP→2118+1=
2119とします。
SP←SP+1
2119
次にWレジスタに2119番地の内容20を入れます。
そしてさらに、SP←SP+1とします。ここで、SPは211Aとなって元に戻ります。
夏子
RAM
20
2119
2119
このあと、211Aとなります。
W
Z
20
1D
PC
201Dこれは、CALL命令の次のOPコードの入っている番地ですね。
SPによってRAMに待避させていた番地を復元させたことになる。
201D
W Z
PC
このWZの中身201Dをプログラムカウンタに入れるとリターン命令の完了です。
カチャ
201D
教科書的なこの図の意味がおわかりいただけたでしょうか。
メインルーチン
サブルーチン
CALL
RET
プログラム作成上何度も使う部分はサブルーチンにしてしまいます。
WAiT
DELAY
Disp
それによって新たに開発する部分との区別がはっきりしますし、
CALL Disp
CALL WAiT
CALL Disp
プログラムのデバッグ作業も楽になるのです。
完成したサブルーチンはデバッグがいりませんからね。
虫
CALL WAiT
虫はいない
虫

持っていますよ。PUSHとPOPの命令が使われた時、レジスタからデータを取り出してメモリに入れたりメモリからレジスタに返したりする仕事です。

PUSH B
⋮
POP B

この他にもSPちゃんは何か仕事を持っていますか？

ありがとう。サブルーチンの働くしくみについてはこれでわかりました。

メインルーチン
E=05
CALL DATA
DATA
E=17
E=E-1
RET
サブルーチン
この時間には、Eレジスタの内容は、05ではなくなっています。

サブルーチンを呼んだ時メインルーチンで使っていたレジスタの内容が変わってしまうことが考えられます。

ホントは、サブルーチンの方に使用するレジスタを守るようにしておくのが礼儀

レジスタは新しいデータがくると古いものは消えてしまいますね。

A
3 6
9
C

2バイトずつ、次の4種類のデータを保護できます。

Aレジスタとフラグ
BCレジスタペア
DE 〃 〃
HL 〃 〃

E=05
データの退避
PUSH D
SP→ E D
CALL DATA
この中でDEレジスタがどう使われようと平気です。
データの復元
POP D
（ホントはサブルーチンプログラムの中に入れておくべきです。）

そこでサブルーチンを呼ぶ前にデータをメモリに退避させ、呼んだ後で元に戻そうというものです。

たとえば、PUSH Bという命令が実行されると、

B・Cレジスタペアの内容がRAMのSP番地に移ります。

SP→ 2118 7 6
2119 5 4
211A
RAM
2118
2119
2118
① SP←SP-1
② 5 4
③ SP←SP-2
④ 7 6
B
C
実行前のSPの内容は211Aとする。

このあと、CALL命令などで、サブルーチンを実行してレジスタB・Cの内容が変わっても、

MOV B,M
MOV B,A

02
F3

RAMの中に、その内容が保持され、番地はSPが記憶しています。

SP ⇨ 2118 76 (C)
2119 54 (B)
211A

そして、POP B命令によってRAMのデータがBCレジスタに戻されます。

SP=2118
① SP+1 =2119
② SP+1=211A

54
76

たとえばこのようなプログラムでは、

① PUSH B
② PUSH PSW
③ PUSH D
④ CALL DATE
⑤

このようにスタックエリアにメモリされていきます。

リターン完了後

PC L
PC H
E
D
フラグ
A
C
B

⑤ ← ④ ← ③ ← ② ← ①

戻す時は順序が逆になります。

⑥ POP D
⑦ POP PSW
⑧ POP B

一度書き込まれた内容はそのまま残っています。

スタックとして使えるメモリはRAMちゃんだけです。

SP

⑧ ← ⑦ ← ⑥

この時わざと戻す順序を変えれば、

PUSH B
PUSH H
POP B
POP H

L
H
C
B

レジスタ間でデータチェンジしたことになります。

だいたい8080の動作についてわかっていただけましたか？ RST、IN、OUTの3つの命令についてはハードウェアのところでくわしくご説明します。

⑩ Z80

日本では6809も大きな勢力を持っていますので、簡単にこの2つのCPUの特徴をお話しましょう。

今8ビットCPUの主流は8080の基本ソフトウェアがそのまま使えるZ80です。

これはソフトでつきあえる8080のレジスタちゃんたちでした。

プログラムカウンタ

スタックポインタ

レジスタ群

Aレジスタとフラグレジスタ

SP B C D E H L A

8080のことがわかっていると、少しのヒントでZ80もわかるはずです。マイクロコンピュータを自在に使いこなすには、CPUのことがわかってないとダメということは何度も痛感しています。

Z80

8080

B'C'D'E'H'L' 3組のレジスタペアとA'F'レジスタ

よろしくね。

Z80ではさらに12のレジスタが8080のそれに追加されました。

F'

B' C' D' E' A' I R IX IY

8080のオペコードと命令ニーモニックを見てください。ニーモニックが…になって空いているコードは10個しかありません。

08
10
18
28
38
CB
D9
DD
ED
FD

30と20は8085で使用している

? 8080は8ビット、Z80も8ビットですからZ80独自のコードはいくつとれるのでしょうか。

命令コードのマップ

	0	1	2	3	…	E	F
0	NOP						RP
1							
2							
3							
:							
F	RRC						

Z80ではこれら22個のレジスタを使う命令があります。また、Z80は8080の命令コードはそのまま使えるという特徴があります。

A	F	A'	F'
B	C	B'	C'
D	E	D'	E'
H	L	H'	L'

I	R
IX	
IY	
SP	
PC	

付加された8ビットでは、256種のコードを表現できます。

Z80

その秘密は、ひとつのコードにさらに8ビットのコードを付加するということにありました。

8080 OPコード

あっ

たった10個のオペコードで、増えた12個のレジスタを含め、どんなソフト機能を追加できるというのでしょうか。

ED

わっ

こうして生まれた新しいコードにZ80のブロック転送・ビット操作などの強力な命令が割り当てられました。8080とコンパチプラスアルファ、これが8080ユーザーにZ80が自然に受け入れられた理由です。

だからZ80専用オペコードは2回読み込む必要があります。1バイト目は8080、2バイト目がZ80で、Z80の命令コードには8080の空きコードを通っていくわけです。

1バイト目 8080コード

2バイト目 Z80コード

FD, ED, DD, D9, CB
38, 28, 18, 10, 08

つまり、8080の10個の空きコードを使って2バイトのオペコードにすれば256×10の命令ニーモニックを割り当てることができます。

そこで、OPコードは同じであるが、ニーモックが違うということに注意する必要があります。つまり8080用アセンブラでは、Z80のニーモニックは使えないということです。

8080アセンブラ

?

?

LD␣A, B

またZ80では8080のニーモック244を変更してZ80用ニーモックにしています。

CPU	ニーモニック	OPコード
Z80	LD␣A, B	78
8080	MOV␣A, B	78

8080では落ちこぼれだったコードがZ80によって一躍スターになったのね。ぐしゅ。よかったわね。

ED DD FD CB

ゼロページの8080のコードを含めFD、ED、DD…の各ページに割り当てられたニーモニックを見てみましょう。

では、まず

0ページ (8080) FD ED DD D9 …

D9 CB DD ED FD

パソコンのCP/M上で走るエディタ、アセンブラ、デバッガーといったプログラムは8080用であることが多いので、その場合Z80CPUは8080CPUを使うのと同じことになります。

まず8080との共通コード以外のものから見ていくことにします。

1バイトのOPコードでZ80の専用になったのは、この8つのコードです。

08 … EX␣AF, AF'
10 … DJNZ, e
18 … JR␣e
20 … JR␣NZ␣e
28 … JR␣Z, e
30 … JR␣NC, e
38 … JR␣C, e
D9 … EXX

では、このうち08、20、D9の3つのOPコードの動作内容を説明します。

08…EX␣AF, AF' これはレジスタ同士の内容交換です。'(ダッシュ)付きレジスタはこの交換以外の役目はありません。

A　A'

フラグレジスタには1ビットごとに役割がある。

20…JR␣NZ, e これは相対ジャンプ命令です。ジャンプの条件はゼロフラグが'0'のときです。

F(フラグレジスタ)
D7　D0
0
"0"
ZERO

この命令の前にゼロフラグを変化させる命令をおき、結果が'0'ならばジャンプさせるようにします。

JR␣NZ, e
1
2
3
e=3

また、eというのはJR␣NZ, eのOPコード20の書かれている番地からの相対的な距離を示しています。

7000 … 20 16 70　JR␣NZ, e
7018 … OPコード1バイト目

ただ、命令の中に直接このeを書き込むことはできません。eだけ離れた番地に書くか、ニーモニックならラベルを使います。

ラベル表記
JR␣NZ, DONE
直接数値
7000　20 16 70
⋮
7018

D9…EXXはBC、DE、HLとBC'、DE'、HL'間でデータの入れ換えをします。CALL文の直前に置いてデータを保護します。

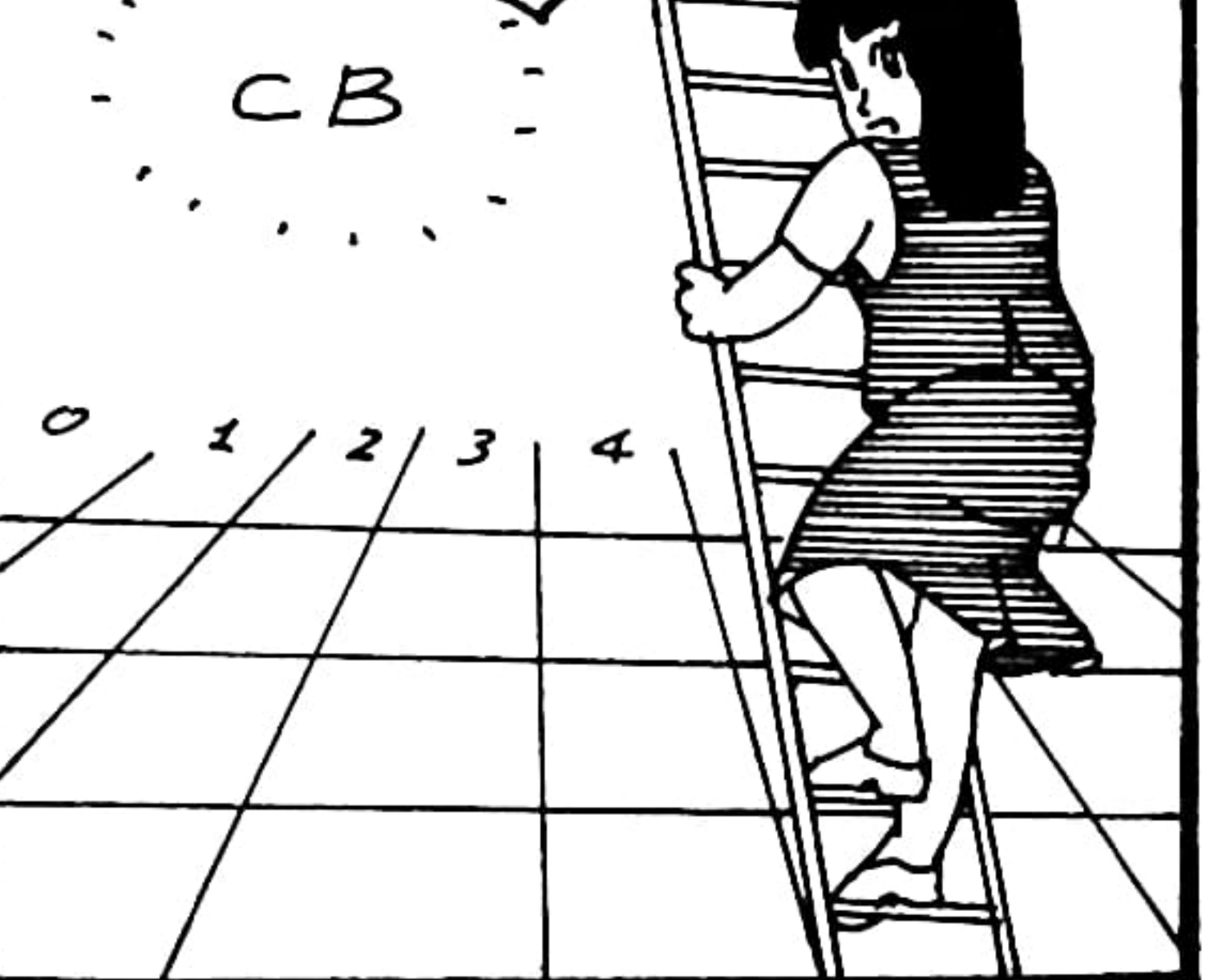
B C D E H L
B' C' D' E' H' L'

POP・PUSHを使うより簡単にできる。

残りのCB、DD、ED、FDの4つのコードが2バイト、3バイトのOPコードへの入口になっています。

CB
0 1 2 3 4

まずCBを1バイト目にもつ命令群を見てみましょう。

0〜3　4〜7　8〜B　C〜F
ビットシフト　ビットテスト　ビットリセット　ビットセット

I/Oポートについてはこの後の8080のハードウェアの所で詳しく説明しています。

メモリのアドレスに相当する

DEV.No.	D7	D6
00	印字押釦	
01		

このビット操作命令はI/Oポートのハンドリングをわかりやすくします。

このCB…で表現されるOPコードはビット操作命令です。

チェック
リセット
セット

C

ビットチェック命令を使うとこうなるの。ニーモックBIT␣7,AによってAレジスタのD7のビットをチェックしようとしていることがわかります。

IN␣A,(00)
BIT␣7,A
Z=1
JP
JP␣Z,n1n2
Z=0

このZ＝1というのがくせものだったわね。結果が0のときがゼロフラグが'1'だけセットされるわけ。

Zフラグ"="1"
A
0000 0000
ZERO
ゼロ！

これがそのチェックルーチンです。

IN␣00
ANI␣80
Z=1
JZ
Z=0

BIT␣7,AはAレジスタのD7のビットをチェックするものです。

D7＝1だからZフラグをおろす

D7 D6 D5 D4
1
ハッ

IN␣A,(00)の内容です。

A
デバイスNo.
00
00ポートのデータをAレジスタに入れます。

こうしてみるとZ80のニーモニックの方が命令の動作内容をよく表現しているように思えます。

Z80　　8080
IN␣A,(00)　　IN␣00

OPコードはどちらもDBです。

(HL)は前にでてきた(M)に当たります。つまり(　)内は番地ポートNo.を示し(　)を含めるとそれは示された番地やポートにあるデータ、ということになります。

(HL)
H　L
3
アドレス
BIT␣1,(HL)

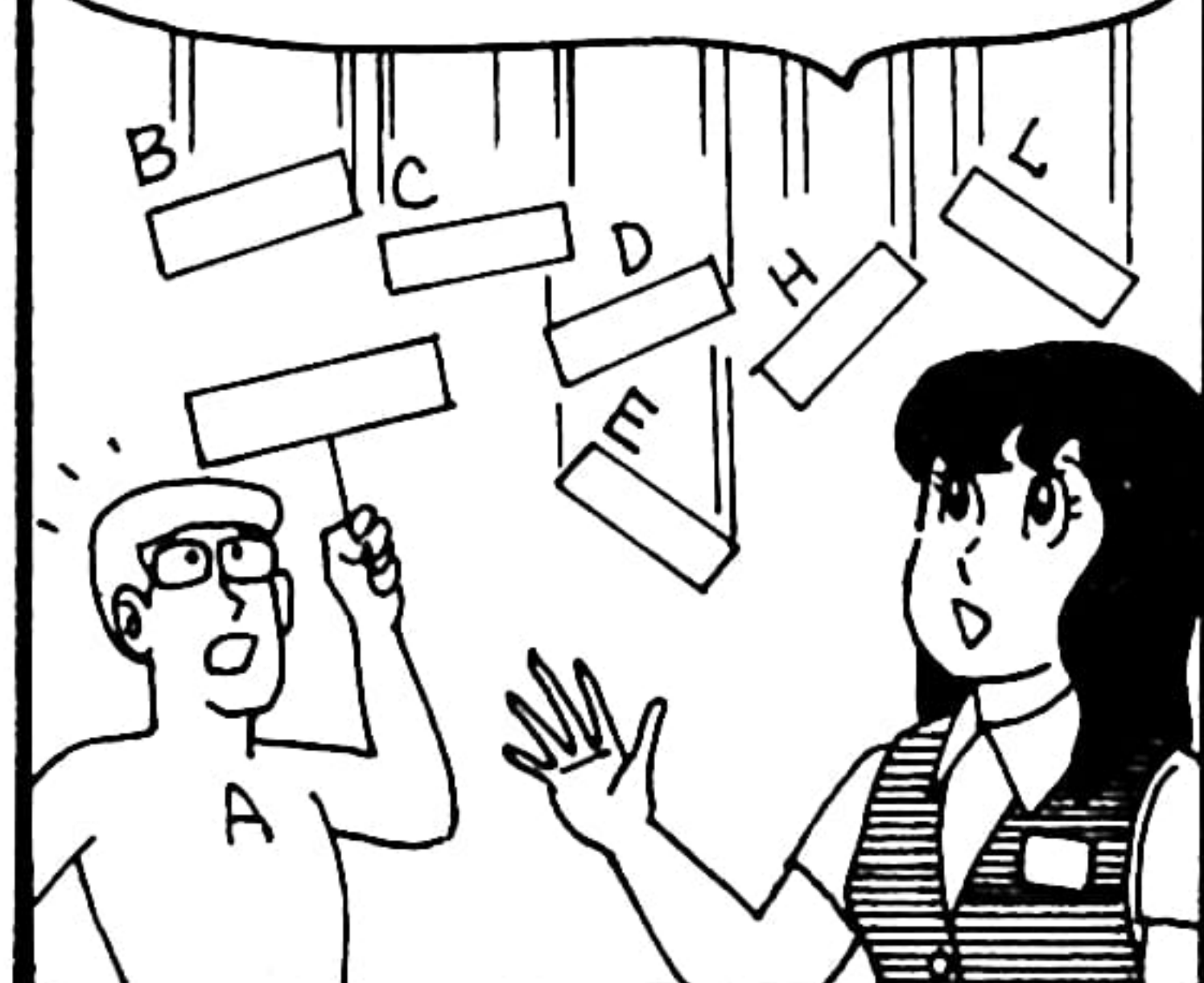
ところでIN␣A,(00)の(　)表現はほかにもBIT␣0,(HL)のように使われていますが、これはどのような意味があるのでしょうか。

HLと(HL)

SET(セット)、RES(リセット)命令もレジスタの中の任意の、ビットをON、OFFするもので、OUT命令と組み合わせて、たとえばパイロットランプ、リレーのON、OFFに応用できます。

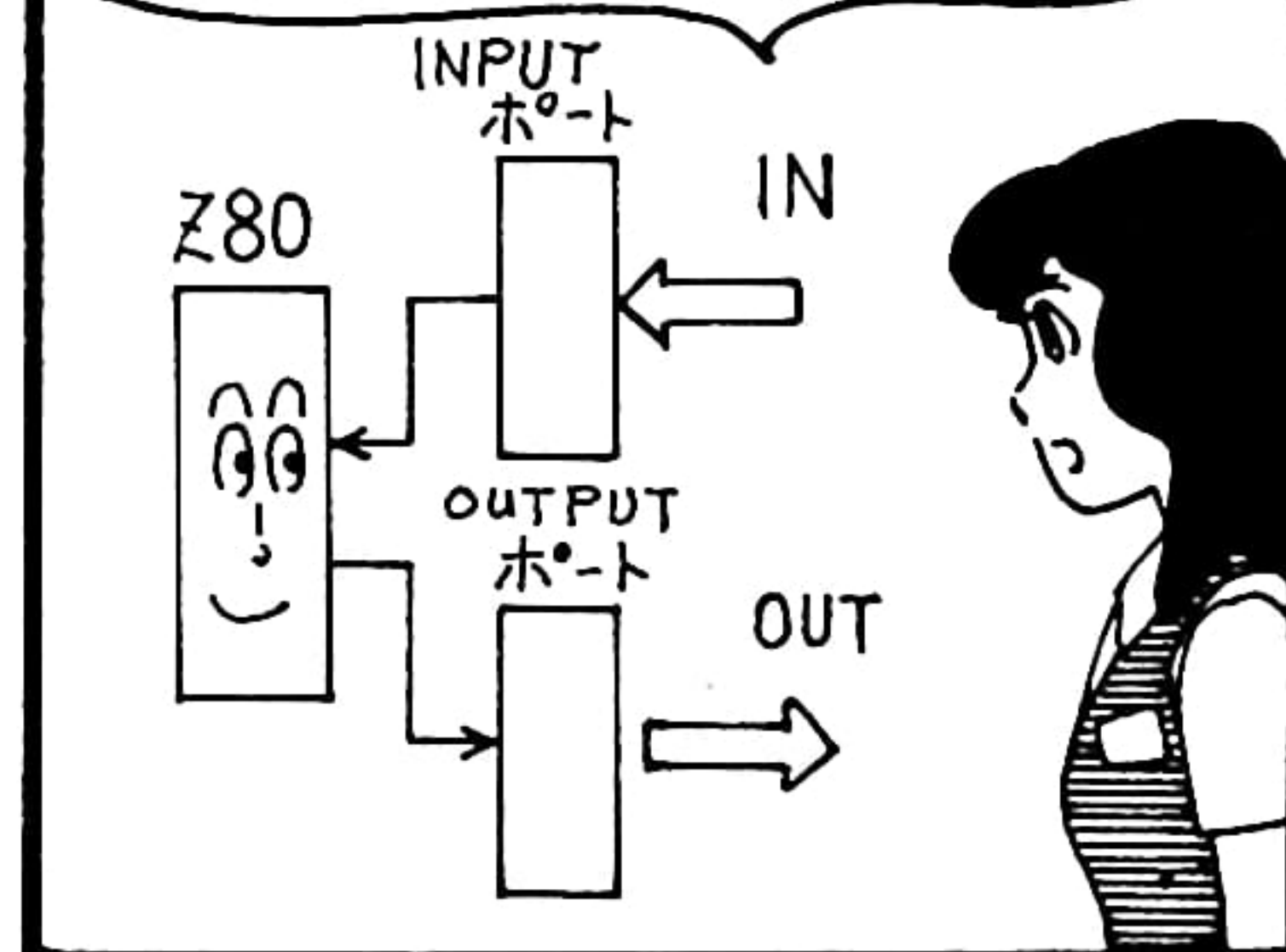
SET␣3,A
OUT␣(01),A
A
D7　D0
1
D7　D0
1

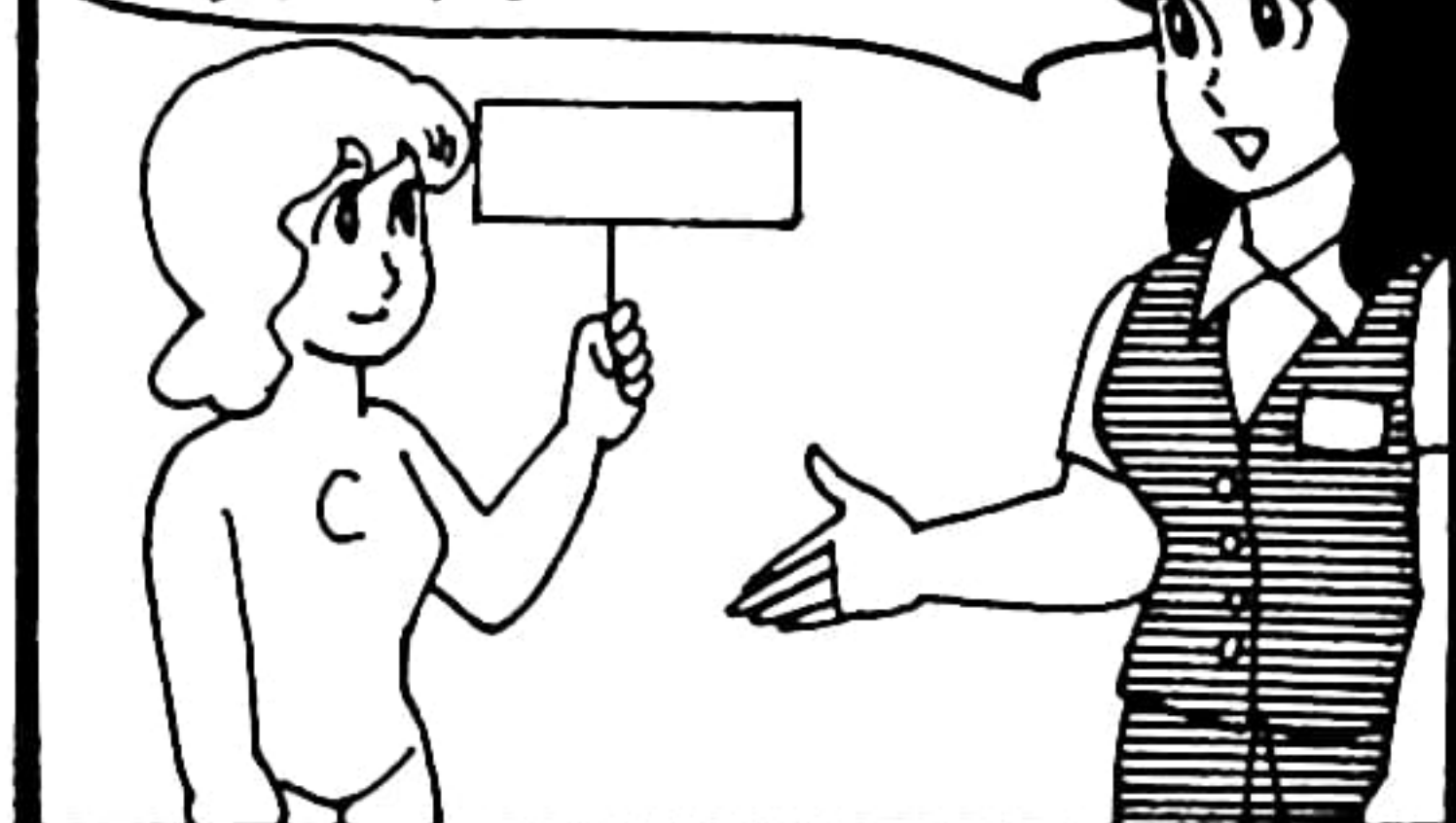
Z80では、さらにB、C、D、E、H、Lの各レジスタを介したI/O命令が追加されています。

B C D H L E A

8080ではI/O命令は、D3のOUT␣D8とDBのIN␣D8の2つのみでした。データはどちらもAレジスタを介して出入りしました。

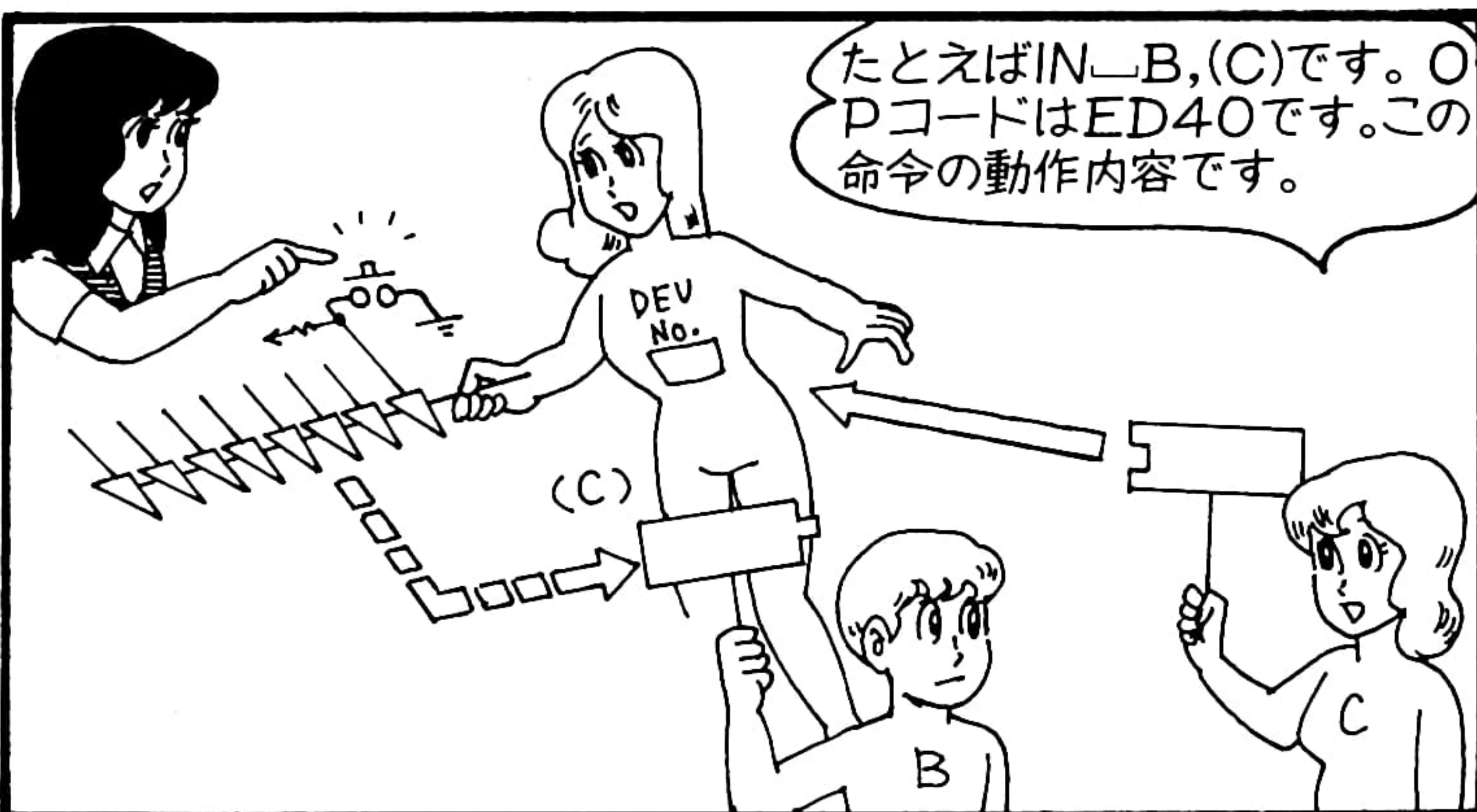
A

あとDD、ED、FDがありますがまずI/O命令に関係する、EDを1バイト目に持つOPコードを見てみます。

INPUT ポート
Z80
IN
OUTPUT ポート
OUT

このED…で表現されるI/O命令の特徴はデバイスNo.の指定がCレジスタで間接的にできるということです。間接的ということは、プログラマの立場では非常に重要です。

C

たとえばIN␣B,(C)です。OPコードはED40です。この命令の動作内容です。

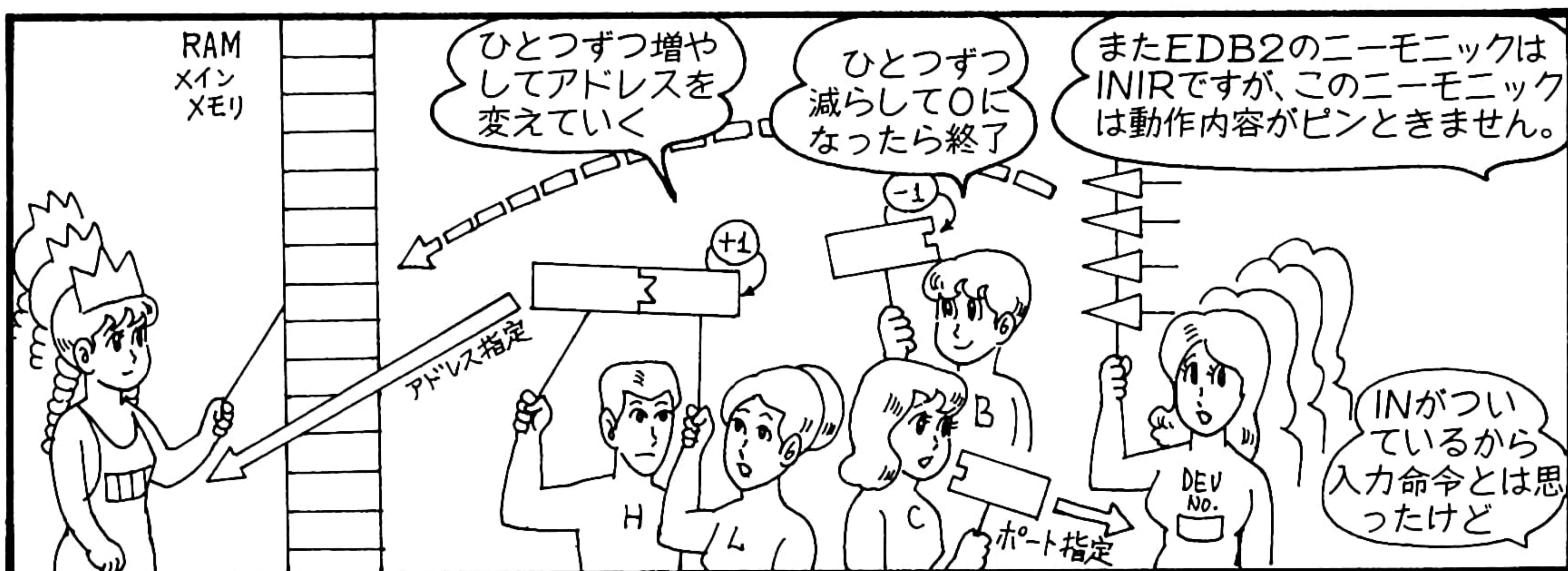
DEV No.
(C)
B
C

RAM
メイン
メモリ

ひとつずつ増やしてアドレスを変えていく

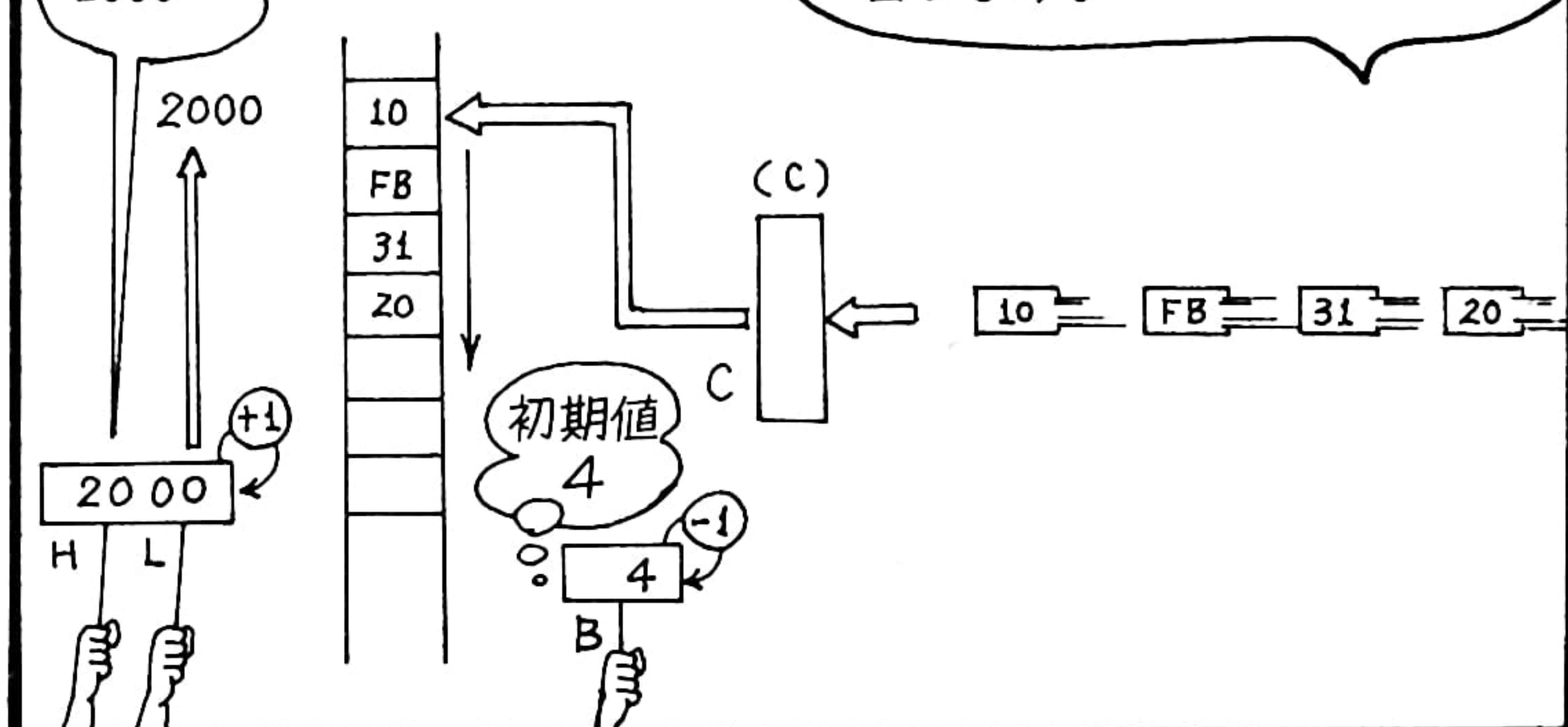
ひとつずつ減らして0になったら終了

またEDB2のニーモニックはINIRですが、このニーモニックは動作内容がピンときません。

+1
-1
アドレス指定
H L C B
ポート指定
DEV No.

INがついているから入力命令とは思ったけど

初期値 2000

名づけてブロック入力命令と言います。

2000
10
FB
31
20
(C)
C
10 FB 31 20
初期値 4
2000
+1
H L
4
-1
B

Cレジスタで指定されるポートから、入ってくるデータをB個HLが指定するHLからHL+B番地までのメモリに書き込むのがこの命令です。

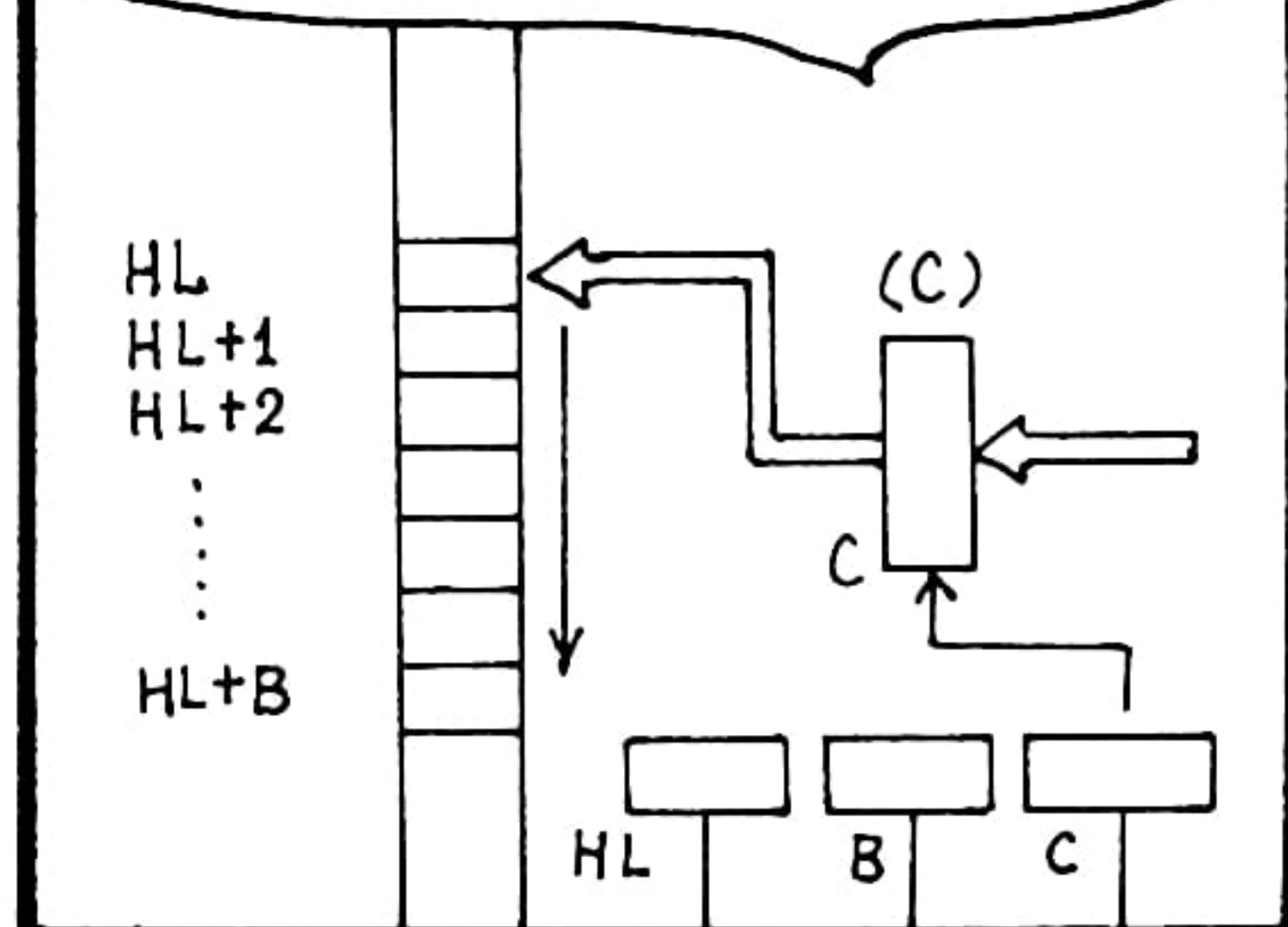
HL
HL+1
HL+2
⋮
HL+B
(C)
C
HL B C

残ったのは、DDとFDですが、これらは、それぞれインデックレジスタＩＸとＩＹに関する命令です。

+d

+d

IX

IY

コードの割当表を見てもらえばわかると思いますが、アドレス指定はIX+d、IY+dのようになっています。

DD 70 05 …. LD␣(IX+5), B

⑤

IX+d

2013

(IX) ⇨ 0

1

2

3

2018をアドレス指定

FF

FF

B

IX

8080ではHLレジスタが、このインデックスレジスタの役目をしていますが +dのような間接でかつ相対的な扱いはできませんでした。

d=0、1、2、…
…FE、FF

ブロック転送

(IX)

+d

さらに、ですね。

DDとFDのCBコードは3バイトOPコードへの入口になっていることがわかりました。内容は(IX+d)と(IY+d)のビット操作命令でした。

DD

FD

CB

d

CB

d

FD CB d ［　］

［　］ここでOPコードが決定

結局8080とコンパチなコードは244個、Z80専用コードは451個ありました。

1バイト8080コンパチCPコード 244個

248個

39個

55個

39個

CB

DD

ED

FD

DD・CB・d

31個

CB

DD ED FD

CB

CB

FD・CB・d

31個

Z80専用 8個

普通Z80のテキストではこんなコードマップでなく、機能別のニーモニック優先の表になっています。

Z-80

Z-80マイコン実習入門

Z-80アセンブラプログラミング

マイクロコンピュータZ80

たとえば、こんな具合です。

8ビットコード

第1オペランド

第2オペランド

OPコード

OPコード	第1オペランド＼第2オペランド	A	B	C	D	E	H	L
LD	A	7F	78	79	7A	7B	7C	7D
	B	47	40	41	42	43	44	45
	C	4F	48	49	4A	4B	4C	4D
	D							

次の6ページはZ80のコード表です。8080 の命令と対比できるようにしました。

LD␣A, A
のマシン語は
7F

Z80コード表 (8080コンパチ)

このページのコード表は8080のコード表と同じです。変な言い方ですがたとえば61というコードはこの表のニーモニックではLD␣H, Cですが8080コード表ではMOV␣H, Cです。Z80も8080もこの61というコードに対して(H)←(C)の動作をします。つまり8080で作ったマシン語プログラムでZ80 CPUは動作するのです。

	0・	1・	2・	3・	4・	5・	6・	7・	8・	9・	A・	B・	C・	D・	E・	F・
・0	NOP	DJNZ␣e	JR␣NZ,e	JR␣NC,e	LD␣B,B	LD␣D,B	LD␣H,B	LD␣(HL),B	ADD␣A,B	SUB␣B	AND␣B	OR␣B	RET␣NZ	RET␣NC	RET␣PO	RET␣P
・1	LD␣BC,n1n2	LD␣DE,n1n2	LD␣HL,n1n2	LD␣SP,n1n2	LD␣B,C	LD␣D,C	LD␣H,C	LD␣(HL),C	ADD␣A,C	SUB␣C	AND␣C	OR␣C	POP␣BC	POP␣DE	POP␣HL	POP␣AF
・2	LD␣(BC),A	LD␣(DE),A	LD␣(n1n2),HL	LD␣(n1n2),A	LD␣B,C	LD␣D,D	LD␣H,D	LD␣(HL),D	ADD␣A,D	SUB␣D	AND␣D	OR␣D	JP␣NZ␣n1n2	JP␣NC,n1n2	JP␣PO␣n1n2	JP␣P␣n1n2
・3	INC␣BC	INC␣DE	INC␣HL	INC␣SP	LD␣B,E	LD␣D,E	LD␣H,E	LD␣(HL),E	ADD␣A,E	SUB␣E	AND␣E	OR␣E	JP␣n1n2	OUT␣(n),A	EX␣(SP),HL	DI
・4	INC␣B	INC␣D	INC␣H	INC␣(HL)	LD␣B,H	LD␣D,H	LD␣H,H	LD␣(HL),H	ADD␣A,H	SUB␣H	AND␣H	OR␣H	CALL NZ,n1n2	CALL␣NC,n1n2	CALL␣PO,n1n2	CALL␣P␣n1n2
・5	DEC␣B	DEC␣D	DEC␣H	DEC␣(HL)	LD␣B,L	LD␣D,L	LD␣H,L	LD␣(HL),L	ADD␣A,L	SUB␣L	AND␣L	OR␣L	PUSH␣BC	PUSH␣DE	PUSH HL	PUSH␣AF
・6	LD␣B,n	LD␣D,n	LD␣H,n	LD␣(HL),n	LD␣B,(HL)	LD␣D,(HL)	LD␣H,(HL)	HALT	ADD␣A,(HL)	SUB␣(HL)	AND␣(HL)	OR␣(HL)	ADD␣A,n	SUB␣n	AND␣n	OR␣n
・7	RLCA	RLA	DAA	SCF	LD␣B,A	LD␣D,A	LD␣H,A	LD␣(HL),A	ADD␣A,A	SUB␣A	AND␣A	OR␣A	RST0	RST2	RST4	RST6
・8	EX␣AF,AF'	JR␣e	JR␣Z,e	JR␣C,e	LD␣C,B	LD␣E,B	LD␣L,B	LD␣A,B	ADC␣A,B	SBC␣A,B	XOR␣B	CP␣B	RET␣Z	RET␣C	RET␣PE	RET␣M
・9	ADD␣HL,BC	ADD␣HL,DE	ADD␣HL,HL	ADD HL,SP	LD␣C,C	LD␣E,C	LD␣L,C	LD␣A,C	ADC␣A,C	SBC␣A,C	XOR␣C	CP␣C	RET	EXX	JP␣(HL)	LD␣SP,HL
・A	LD␣A,(BC)	LD␣A,(DE)	LD␣HL,(n1n2)	LD␣A,(n1n2)	LD␣C,D	LD␣E,D	LD␣L,D	LD␣A,D	ADC␣A,D	SBC␣A,D	XOR␣D	CP␣D	JP␣Z,n1n2	JP␣C,n1n2	JP␣PE,n1n2	JP␣M,n1n2
・B	DEC␣BC	DEC␣DE	DEC␣HL	DEC␣SP	LD␣C,E	LD␣E,E	LD␣L,E	LD␣A,E	ADC␣A,E	SBC␣A,E	XOR␣E	CP␣E	CBページ	IN␣A,(n)	EX␣ED,HL	EI
・C	INC␣C	INC␣E	INC␣L	INC␣A	LD␣C,H	LD␣E,H	LD␣L,H	LD␣A,H	ADC␣A,H	SBC␣A,H	XOR␣H	CP␣H	CALL␣NZ,n1n2	CALL␣C␣n1n2	CALL␣PE,n1n2	CALL␣M,n1n2
・D	DEC␣C	DEC␣E	DEC␣L	DEC␣A	LD␣C,L	LD␣E,L	LD␣L,L	LD␣A,L	ADC␣A,L	SBC␣A,L	XOR␣L	CP␣L	CALL␣n1n2	DDページ	EDページ	FDページ
・E	LD␣C,n	LD␣E,n	LD␣L,n	LD␣A,n	LD␣C,(HL)	LD␣E,(HL)	LD␣L,(HL)	LD␣A,(HL)	ADC␣A,(HL)	SBC␣A,(HL)	XOR␣(HL)	CP␣(HL)	ADC␣A,n	SBC␣A,n	XOR␣n	CP␣n
・F	RRCA	RRA	CPL	CCF	LD␣C,A	LD␣E,A	LD␣L,A	LD␣A,A	ADC␣A,A	SBC␣A,A	XOR␣A	CP␣A	RST1	RST3	RST5	RST7

（RST0・RST2・RST4・RST6 および RST1・RST3・RST5・RST7：←このコードはハード的にデータバスに入れる→）

Z80 CPUのニーモニック-OPコード割当表(Z80専用コード)

CBページ

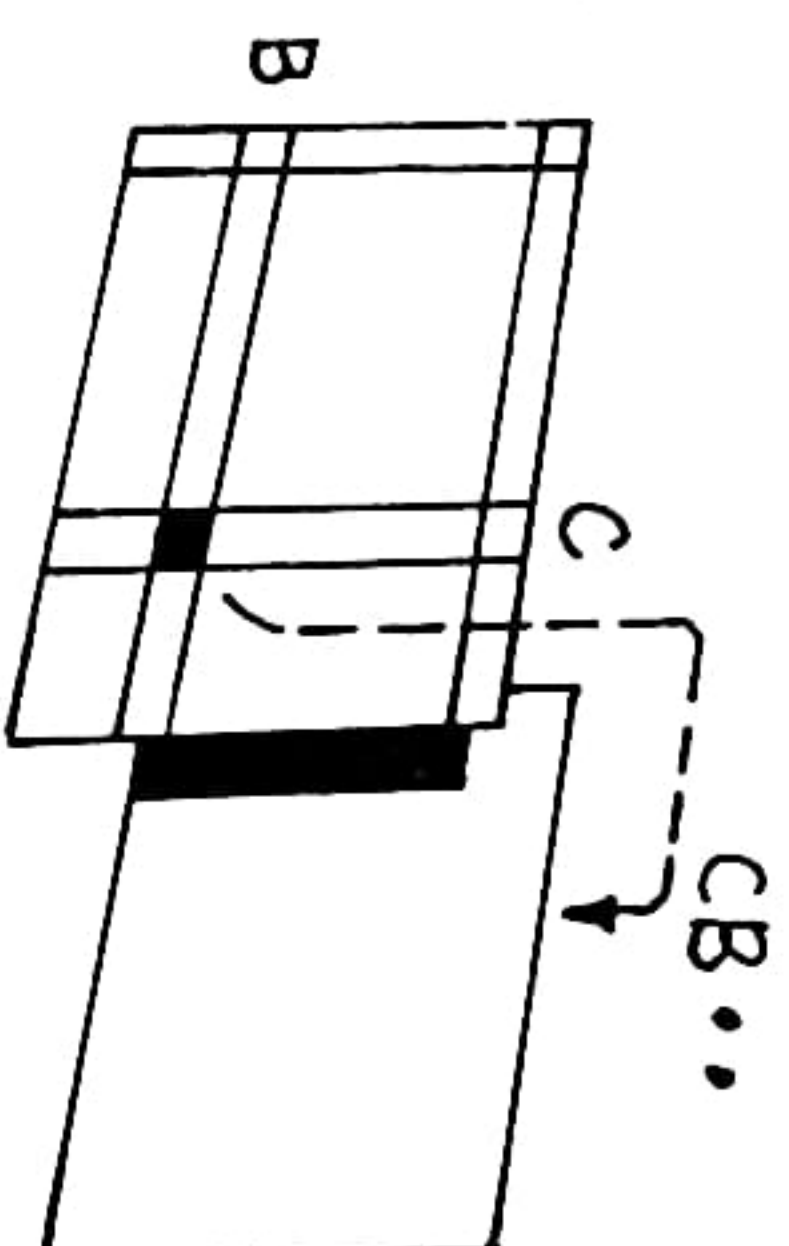
この表にあるニーモニックのOPコードはすべて2バイトです。1バイト目がCB 2バイト目がこの表のコードです。

たとえば、BIT␣6,BはCB 70というOPコードになります。

CB・・

C

B

2バイト

	0・	1・	2・	3・	4・	5・	6・	7・	8・	9・	A・	B・	C・	D・	E・	F・
・0	RLC␣B	RL␣B	SLA␣B		BIT␣0,B	BIT␣2,B	BIT␣4,B	BIT␣6,B	RES␣0,B	RES␣2,B	RES␣4,B	RES␣6,B	SET␣0,B	SET␣2,B	SET␣4,B	SET␣6,B
・1	RLC␣C	RL␣C	SLA␣C		BIT␣0,C	BIT␣2,C	BIT␣4,C	BIT␣6,C	RES␣0,C	RES␣2,C	RES␣4,C	RES␣6,C	SET␣0,C	SET␣2,C	SET␣4,C	SET␣6,C
・2	RLC␣D	RL␣D	SLA␣D		BIT␣0,D	BIT␣2,D	BIT␣4,D	BIT 6,D	RES␣0,D	RES␣2,D	RES␣4,D	RES␣6,D	SET␣0,D	SET␣2,D	SET␣4,D	SET␣6,D
・3	RLC␣E	RL␣E	SLA␣E		BIT␣0,E	BIT␣2,E	BIT␣4,E	BIT 6,E	RES␣0,E	RES␣2,E	RES␣4,E	RES␣6,E	SET␣0,E	SET␣2,E	SET␣4,E	SET␣6,E
・4	RLC␣H	RL␣H	SLA␣H		BIT␣0,H	BIT␣2,H	BIT␣4,H	BIT 6,H	RES␣0,H	RES␣2,H	RES␣4,H	RES␣6,H	SET␣0,H	SET␣2,H	SET␣4,H	SET␣6,H
・5	RLC␣L	RL␣L	SLA␣L		BIT␣0,L	BIT␣2,L	BIT␣4,L	BIT 6,L	RES␣0,L	RES␣2,L	RES␣4,L	RES␣6,L	SET␣0,L	SET␣2,L	SET␣4,L	SET␣6,L
・6	RLC␣(HL)	RL␣(HL)	SLA␣(HL)		BIT␣0,(HL)	BIT␣2,(HL)	BIT␣4,(HL)	BIT 6,(HL)	RES␣0,(HL)	RES␣2,(HL)	RES␣4,(HL)	RES␣6,(HL)	SET␣0,(HL)	SET␣2,(HL)	SET␣4,(HL)	SET␣6,(HL)
・7	RLC␣A	RL␣A	SLA␣A		BIT␣0,A	BIT␣2,A	BIT␣4,A	BIT␣6,A	RES␣0,A	RES␣2,A	RES␣4,A	RES␣6,A	SET␣0,A	SET␣2,A	SET␣4,A	SET␣6,A
・8	RRC␣B	RR␣B	SRA␣B	SRL␣B	BIT␣1,B	BIT␣3,B	BIT␣5,B	BIT␣7,B	RES␣1,B	RES␣3,B	RES␣5,B	RES␣7,B	SET␣1,B	SET␣3,B	SET␣5,B	SET␣7,B
・9	RRC␣C	RR␣C	SRA␣C	SRL␣C	BIT␣1,C	BIT␣3,C	BIT␣5,C	BIT␣7,C	RES␣1,C	RES␣3,C	RES␣5,C	RES␣7,C	SET␣1,C	SET␣3,C	SET␣5,C	SET␣7,C
・A	RRC␣D	RR␣D	SRA␣D	SRL␣D	BIT␣1,D	BIT␣3,D	BIT␣5,D	BIT␣7,D	RES␣1,D	RES␣3,D	RES␣5,D	RES␣7,D	SET␣1,D	SET␣3,D	SET␣5,D	SET␣7,D
・B	RRC␣E	RR␣E	SRA␣E	SRL␣E	BIT␣1,E	BIT␣3,E	BIT␣5,E	BIT␣7,E	RES␣1,E	RES␣3,E	RES␣5,E	RES␣7,E	SET␣1,E	SET␣3,E	SET␣5,E	SET␣7,E
・C	RRC␣H	RR␣H	SRA␣H	SRL␣H	BIT␣1,H	BIT␣3,H	BIT␣5,H	BIT␣7,H	RES␣1,H	RES␣3,H	RES␣5,H	RES␣7,H	SET␣1,H	SET␣3,H	SET␣5,H	SET␣7,H
・D	RRC␣L	RR␣L	SRA␣L	SRL␣L	BIT␣1,L	BIT␣3,L	BIT␣5,L	BIT␣7,L	RES␣1,L	RES␣3,L	RES␣5,L	RES␣7,L	SET␣1,L	SET␣3,L	SET␣5,L	SET␣7,L
・E	RRC␣(HL)	RR␣(HL)	SRA␣(HL)	SRL␣(HL)	BIT␣1,(HL)	BIT␣3,(HL)	BIT␣5,(HL)	BIT␣7,(HL)	RES␣1,(HL)	RES␣3,(HL)	RES␣5,(HL)	RES␣7,(HL)	SET␣1,(HL)	SET␣3,(HL)	SET␣5,(HL)	SET␣7,(HL)
・F	RRC␣A	RR␣A	SRA␣A	SRL␣A	BIT␣1,A	BIT␣3,A	BIT␣5,A	BIT␣7,A	RES␣1,A	RES␣3,A	RES 5,A	RES␣7,A	SET␣1,A	SET␣3,A	SET␣5,A	SET␣7,A

Z80 CPUのニーモニック-OPコード割当表(Z80専用コード)

DDページ

このページはインデックスレジスタXの操作命令です。

CBコードはさらに次のページへの入口です。

ニーモニック LD␣IX,n1n2の OPコードは DD21n2n1です。

IX

	0・	1・	2・	3・	4・	5・	6・	7・	8・	9・	A・	B・	C・	D・	E・	F・
・0								LD␣(IX+d),B								
・1			LD␣IX,n1n2					LD␣(IX+d),C							POP␣IX	
・2			LD␣(n1n2),IX					LD␣(IX+d),D								
・3			INC␣IX					LD␣(IX+d),E							EX␣(SP),IX	
・4				INC␣(IX+d)				LD␣(IX+d),H								
・5				DEC␣(IX+d)				LD␣(IX+d),L							PUSH␣IX	
・6				LD␣(IX+d),n	LD␣B,(IX+d)	LD␣D,(IX+d)	LD␣H,(IX+d)		ADD␣A,(IX+d)	SUB␣(IX+d)	AND␣(IX+d)	OR␣(IX+d)				
・7								LD␣(IX+d),A								
・8																
・9	ADD␣IX,BC	ADD␣IX,DE	ADD␣IX,IX	ADD␣IX,SP											JP␣(IX)	LD␣SP,IX
・A			LD␣IX,(n1n2)													
・B			DEC␣IX													
・C																
・D																
・E					LD␣C,(IX+d)	LD␣E,(IX+d)	LD␣L,(IX+d)	LD␣A,(IX+d)	ADC␣A,(IX+d)	SBC␣A,(IX+d)	XOR␣(IX+d)	CP␣(IX+d)				
・F																

Z80 CPUのニーモニック-OPコード割当表
EDページ（Z80専用コード）

	0・	1・	2・	3・	4・	5・	6・	7・	8・	9・	A・	B・	C・	D・	E・	F・
・0					IN␣B,(C)	IN␣D,(C)	IN␣H,(C)				LDI	LDIR				
・1					OUT␣(C),B	OUT (C),D	OUT␣(C),H				CPI	CPIR				
・2					SBC␣HL,BC	SBC␣HL,DE	SBC HL,HL	SBC HL,SP			INI	INIR				
・3					LD␣(n1n2),BC	LD␣(n1n2),DE		LD␣(n1n2),SP			OUTI	OTIR			EX␣(SP),IX	
・4					NEG											
・5																
・6					IM0	IM1										
・7					LD␣I,A	LD␣A,I	RRD␣(HL)									
・8					IN␣C,(C)	IN␣E,(C)	IN␣L,(C)	IN␣A,(C)			LDD	LDDR				
・9					OUT␣(C),C	OUT␣(C),E	OUT␣(C),L	OUT␣(C),A			CPD	CPDR				
・A					ADC␣HL,BC	ADC␣HL,DE	ADC␣HL,HL	ADC␣HL,SP			IND	INDR				
・B					LD␣BC,(n1n2)	LD␣DE,(n1n2)		LD␣SP,(n1n2)			OUTD	OTDR				
・C																
・D																
・E						IM2										
・F					LD␣R,A	LD␣A,R	RLD␣(HL)									

Z80 CPUニーモニック-OPコード割当表(Z80専用コード)

FDページ

このFDページはインデックスレジスタYの操作命令です。

CBコードはさらに次のページへの入口です。

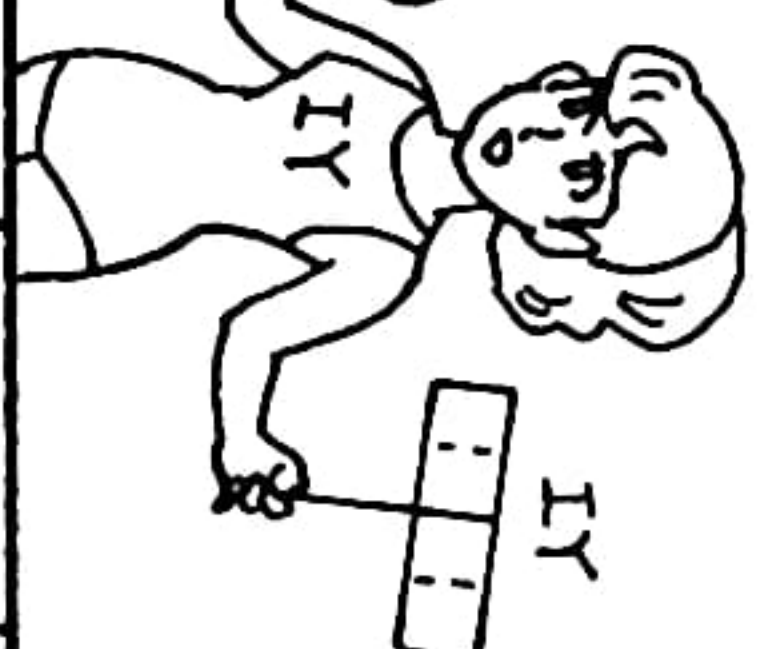
IYはアドレスではありません。

IY

IY+d

(IY+d)

メモリとレジスタは区別してください。

	0・	1・	2・	3・	4・	5・	6・	7・	8・	9・	A・	B・	C・	D・	E・	F・
・0								LD␣ (IY+d),B								
・1			LD␣ IY,n1n2					LD␣ (IY+d),C							POP␣IY	
・2			LD␣ (n1n2),IY					LD␣ (IY+d),D								
・3			INC␣IY					LD␣ (IY+d),E							EX␣ (SP),IY	
・4				INC␣ (IY+d)				LD␣ (IY+d),H								
・5				DEC␣ (IY+d)				LD␣ (IY+d),L							PUSH␣IY	
・6				LD␣ (IY+d),n	LD␣B, (IY+d)	LD␣D, (IY+d)	LD␣H, (IY+d)		ADD␣ A,(IY+d)	SUB␣ (IY+d)	AND␣ (IY+d)	OR␣(IY+d)				
・7								LD␣ (IY+d),A								
・8																
・9	ADD␣ IY,BC	ADD␣ IY,DE	ADD␣ IY,IY	ADD␣ IY,SP											JP␣(IY)	LD␣ SP,IY
・A			LD␣ IY,(n1n2)													
・B			DEC␣IY													
・C																
・D																
・E					LD␣C, (IY+d)	LD␣E, (IY+d)	LD␣L, (IY+d)	LD␣A, (IY+d)	ADC␣ A,(IY+d)	SBC␣ A,(IY+d)	XOR␣ (IY+d)	CP␣(IY+d)				
・F																

Z80 CPUのニーモニック-OPコード割当表(Z80専用コード)

DD CB d ページ

	0・	1・	2・	3・	4・	5・	6・	7・	8・	9・	A・	B・	C・	D・	E・	F・
・6	RLC␣(IX+d)	RL␣(IX+d)	SLA␣(IX+d)	SRL␣(IX+d)	BIT␣0,(IX+d)	BIT␣2,(IX+d)	BIT␣4,(IX+d)	BIT␣6,(IX+d)	RES␣0,(IX+d)	RES␣2,(IX+d)	RES␣4,(IX+d)	RES␣6,(IX+d)	SET␣0,(IX+d)	SET␣2,(IX+d)	SET␣4,(IX+d)	SET␣6,(IX+d)
・E	RRC␣(IX+d)	RR␣(IX+d)	SRA␣(IX+d)		BIT␣1,(IX+d)	BIT␣3,(IX+d)	BIT␣5,(IX+d)	BIT␣7,(IX+d)	RES␣1,(IX+d)	RES␣3,(IX+d)	RES 5,(IX+d)	RES␣7,(IX+d)	SET␣1,(IX+d)	SET␣3,(IX+d)	SET␣5,(IX+d)	SET␣7,(IX+d)

この2つのページのニーモニックの命令は3バイトのOPコードになります。

たとえばニーモニック
SET␣4,(IX+d)
のオペコードは
DDCBdE6になります。

1バイト目
DD
2バイト目
CB
3バイト目 は dが入る
4バイト目

FD CB d ページ

	0・	1・	2・	3・	4・	5・	6・	7・	8・	9・	A・	B・	C・	D・	E・	F・
・6	RLC␣(IY+d)	RL␣(IY+d)	SLA␣(IY+d)	SRL␣(IY+d)	BIT␣0,(IY+d)	BIT␣2,(IY+d)	BIT␣4,(IY+d)	BIT␣6,(IY+d)	RES␣0,(IY+d)	RES␣2,(IY+d)	RES␣4,(IY+d)	RES␣6,(IY+d)	SET␣0,(IY+d)	SET␣2,(IY+d)	SET␣4,(IY+d)	SET␣6,(IY+d)
・E	RRC␣(IY+d)	RR␣(IY+d)	SRA␣(IY+d)		BIT 1,(IY+d)	BIT␣3,(IY+d)	BIT␣5,(IY+d)	BIT␣7,(IY+d)	RES␣1,(IY+d)	RES␣3,(IY+d)	RES␣5,(IY+d)	RES␣7,(IY+d)	SET␣1,(IY+d)	SET␣3,(IY+d)	SET␣5,(IY+d)	SET␣7,(IY+d)

⑫6502と6809

	レジスタ	
	A	8ビット
	X	〃
	Y	〃
PC(H)	PC(L)	16ビット
01	SP	8ビット
	F/F	〃

6502 CPU内部レジスタ

命令数は全部で151個と8080より少ないですね。

6502はモステクノロジー社の製品でワンボードブームの最中にパソコン用に作られたCPUです。

apple

6809 CPU内部レジスタ

X		16ビット
Y		〃
SP(U)		〃
SP(S)		〃
PC		〃
A	B	8ビット×2
	DP	8ビット
	F/F	〃

6809はZ80と双璧を成しているCPUです。

6809はモトローラ社のCPUです。日本では富士通や日立のパソコンなどに採用されています。

まず、レジスタの数がZ80にくらべると少ないですね。たったこれだけのレジスタで8ビットの究極と言われる命令群を持っているのです。そこでZ80の考え方の延長では理解できない部分がでてきます。

そこで6809についても少しだけその特徴的な所をお話してみたいと思います。

1464命令

6809

基本ソフトウェアの充実からZ80が優位を保っていますが、やがて6809の基本ソフトウェアが充実してくればそのシェアは変わっていくでしょう。

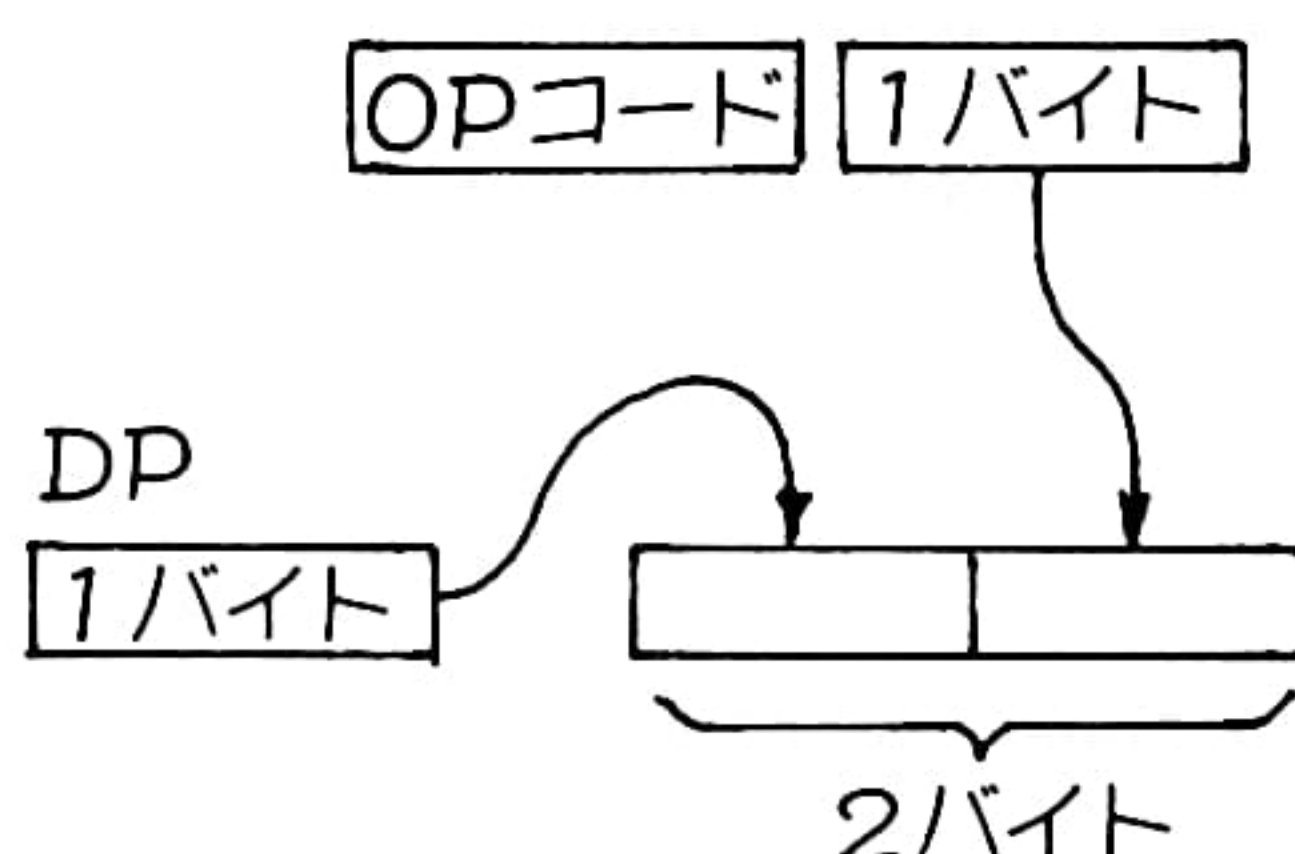
DPはダイレクトページレジスタでアドレスの上桁をこれで指定するようにしたものです。

プログラム

OPコード　1バイト

DP

1バイト

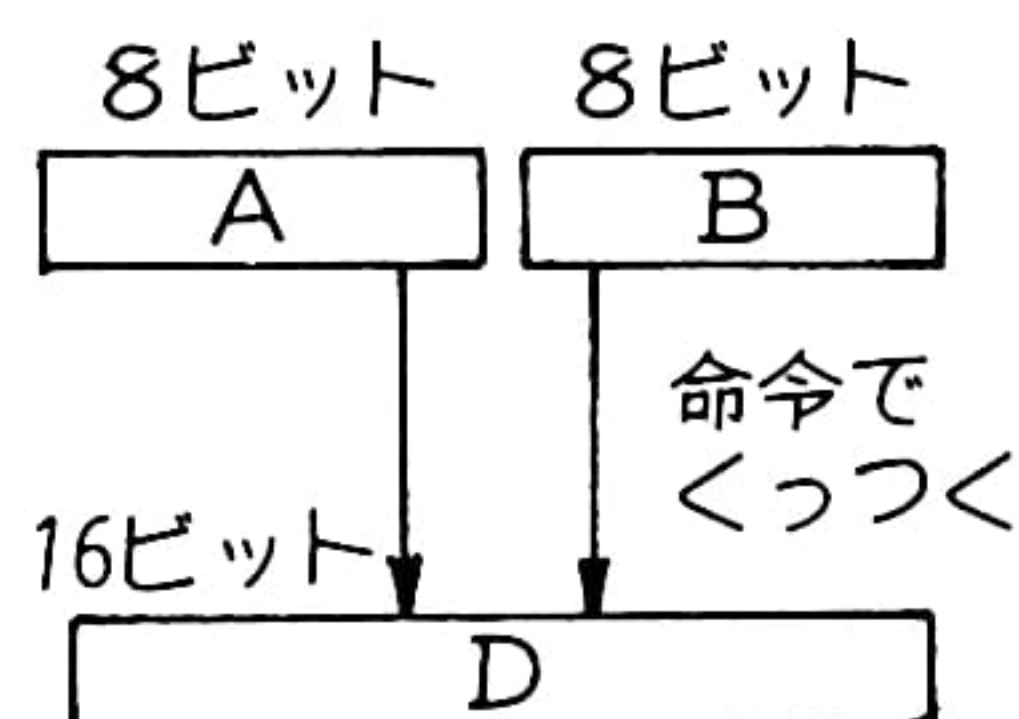
2バイト

次に変わっているのは、アキュムレータがAとBの2つありこれがくっつくとDという16ビットのアキュムレータになるということです。

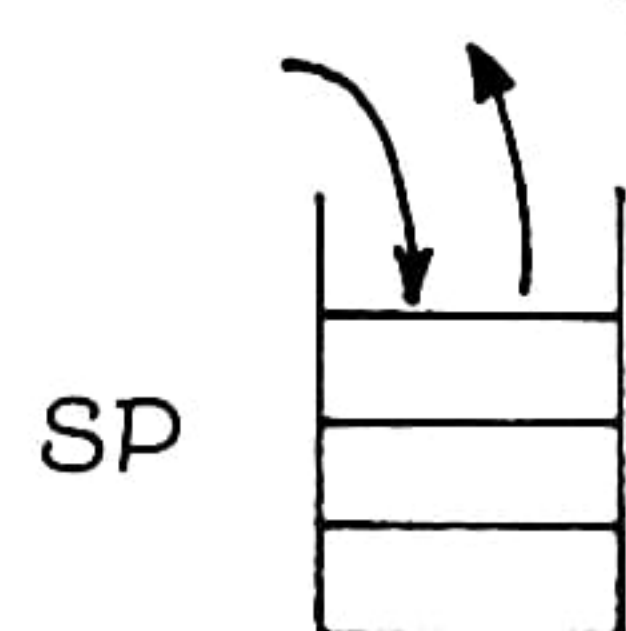
8ビット　8ビット

A　B

命令でくっつく

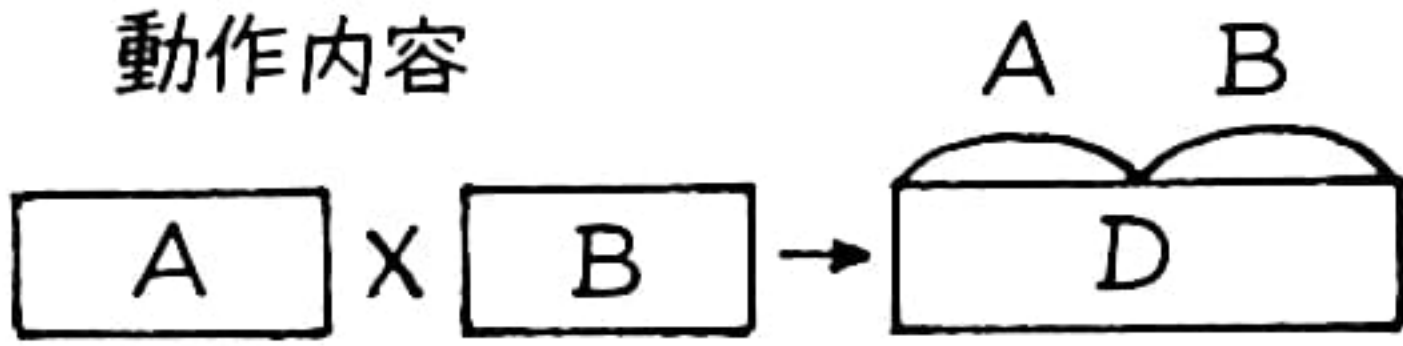
16ビット

D

まずスタックポインタが2つあり、ひとつはシステム用、ひとつはユーザー用となっています。スタックポインタの役目はZ80と変わらないのでこれ以上お話しません。

SP

インハラントは1バイト命令です。レジスタ操作命令です。

ニーモニック	OPコード
MUL	3D

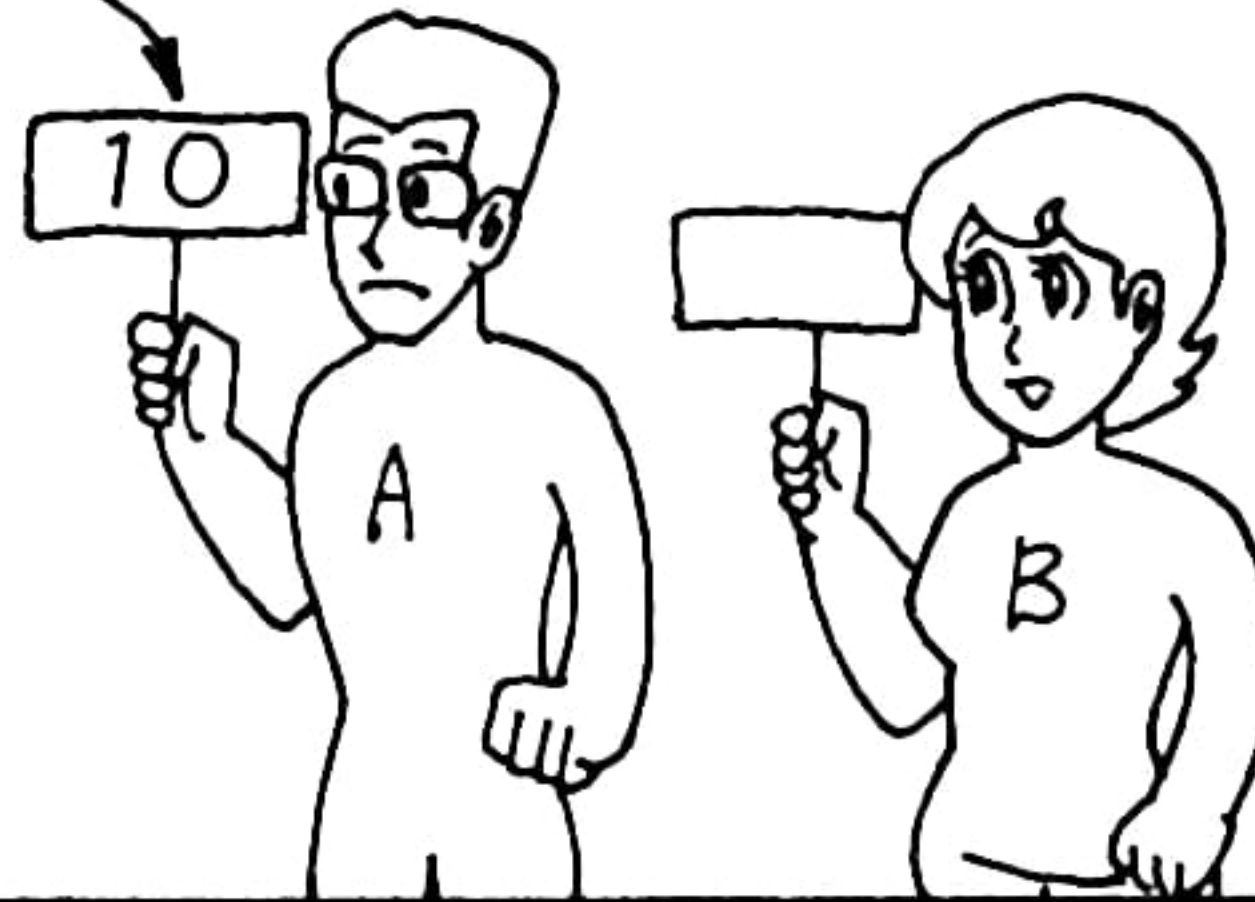
動作内容

A × B → D（A B）

でも、このうちのいくつかはZ80と同じはずですから中身がわかれば新しく覚えなければならないものはひとつかふたつです。

初心者にとって一番イヤラシイのがアドレッシングモードの種類がたくさんあることです。

- インハラント
- イミディエイト
- ダイレクト
- エクステンド
- インデックス
  - インデックス
  - インデックス インダイレクト
  - エクステンド インダイレクト

ニーモニック	OPコード	動作内容
LDA␣#10	8610	M→A 定数をAに入れる。

この#記号がイミディエイトモードであることを示す。したがってOPコードは86となる。

イミディエイトはオペランドに直接数値を書く命令です。ニーモニックでは#記号をつけ他のモードと区別します。

10

A

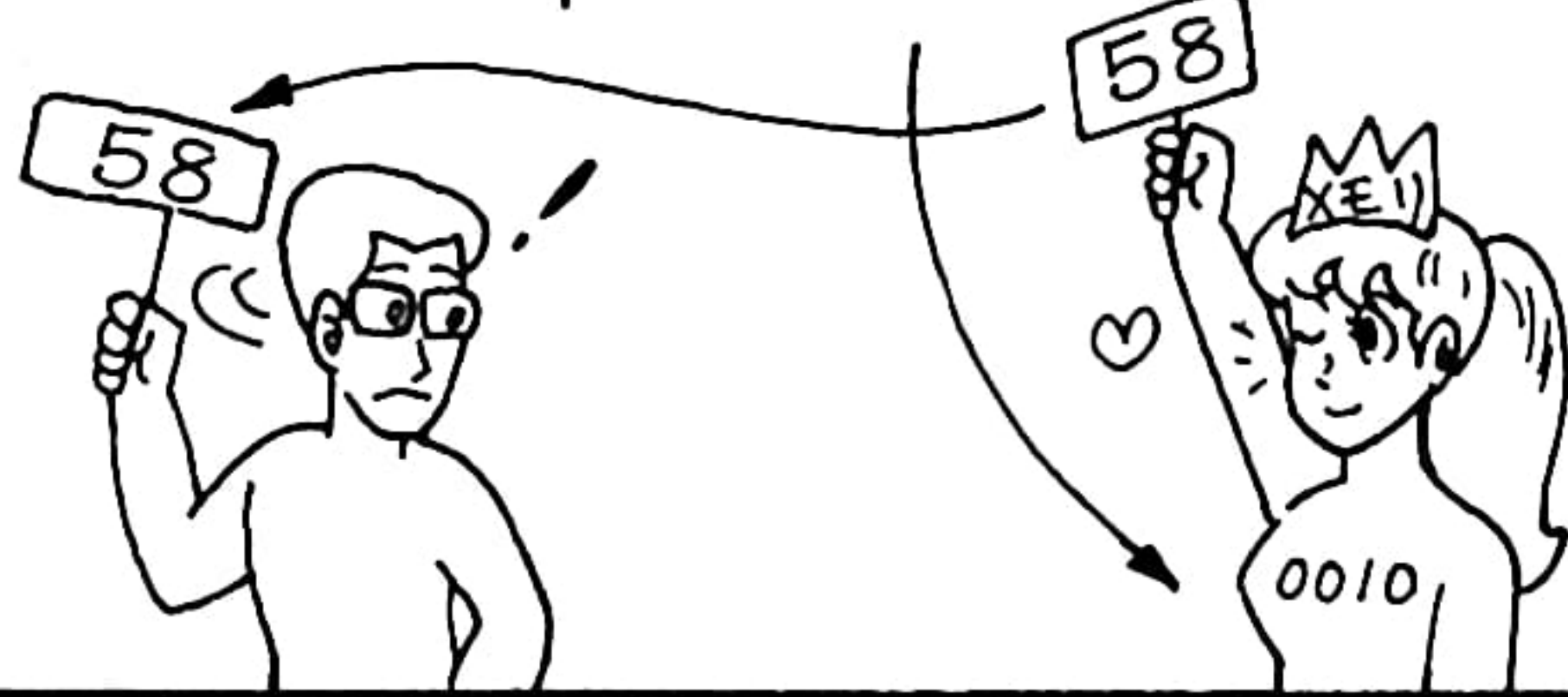
B

次のエクステンドは間接アドレス指定です。>記号で区別されます。

ニーモニック	OPコード	動作内容
LDA␣>10	B6 00 10	M→A

58

58

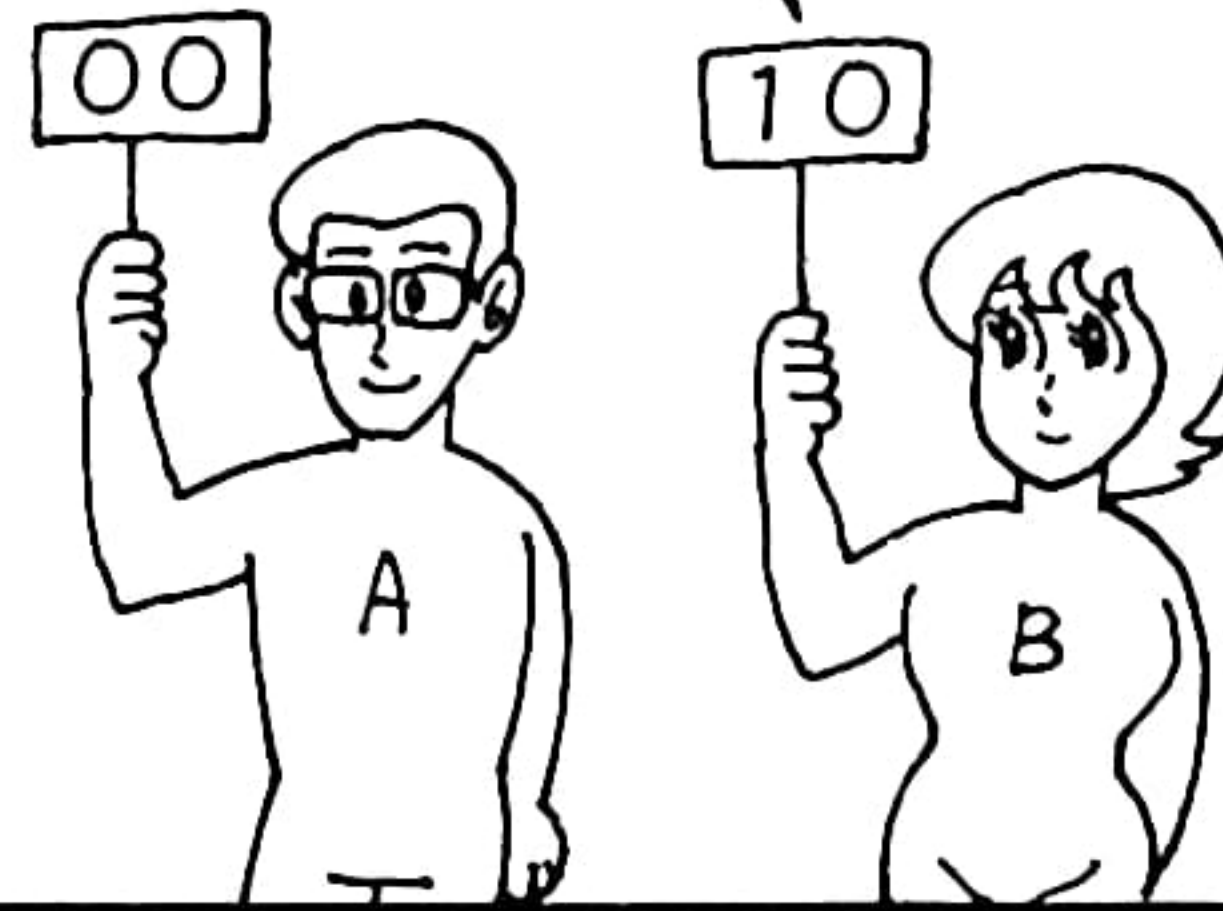
メモリ

0010

ニーモニック	OPコード	内容
LDD␣#0010	CC 0010	M→D

00

10

A

B

Z80の時はニーモニックをマシン語に直すとアドレスの上桁下桁が入れかわりましたが6809ではそのままです。

ニーモニック	OPコード	動作内容
LDD␣>10	FC 0010	M→D

34

12

34

12

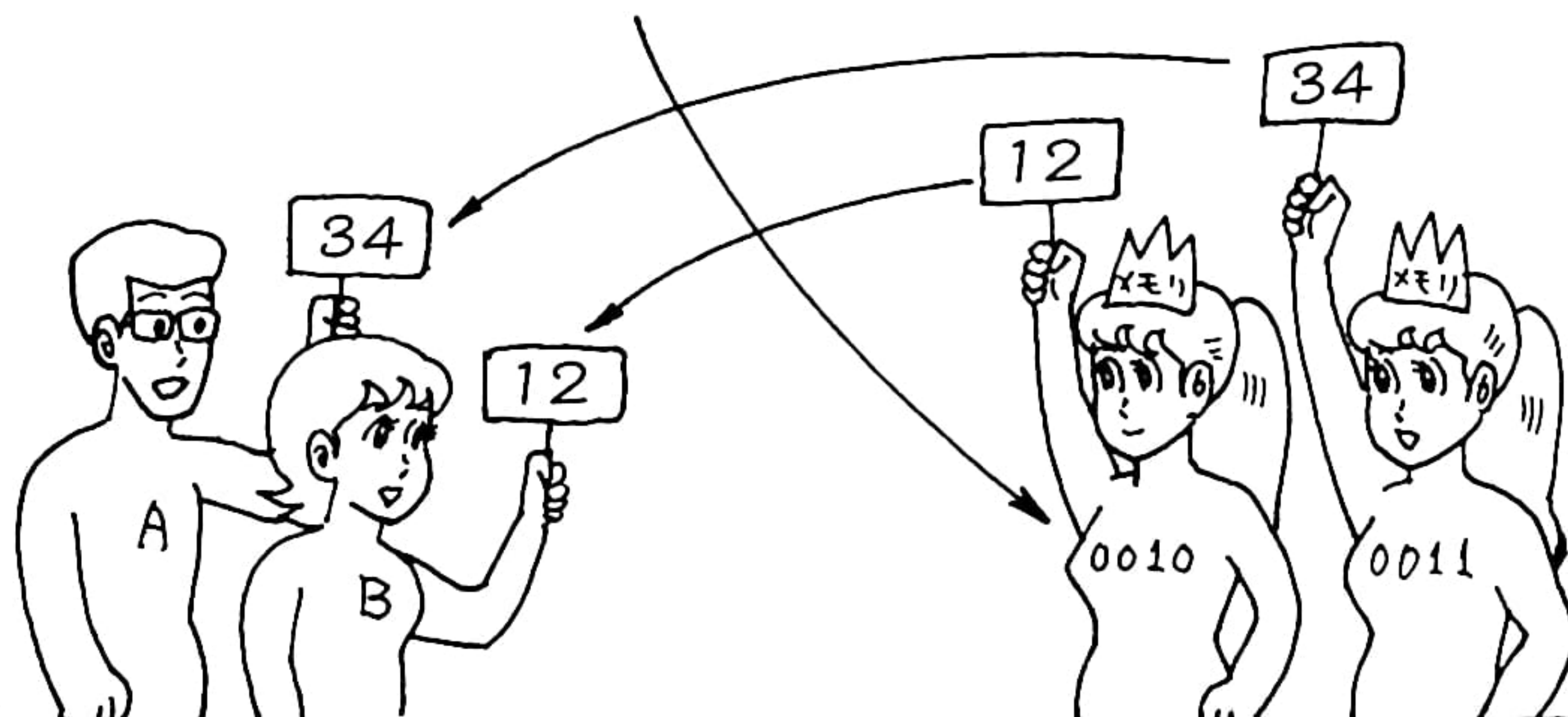
メモリ

メモリ

A

B

0010

0011

次のインデックスモードが非常に難解に思えたモードです。バイト数の後に＋記号がついているのです。

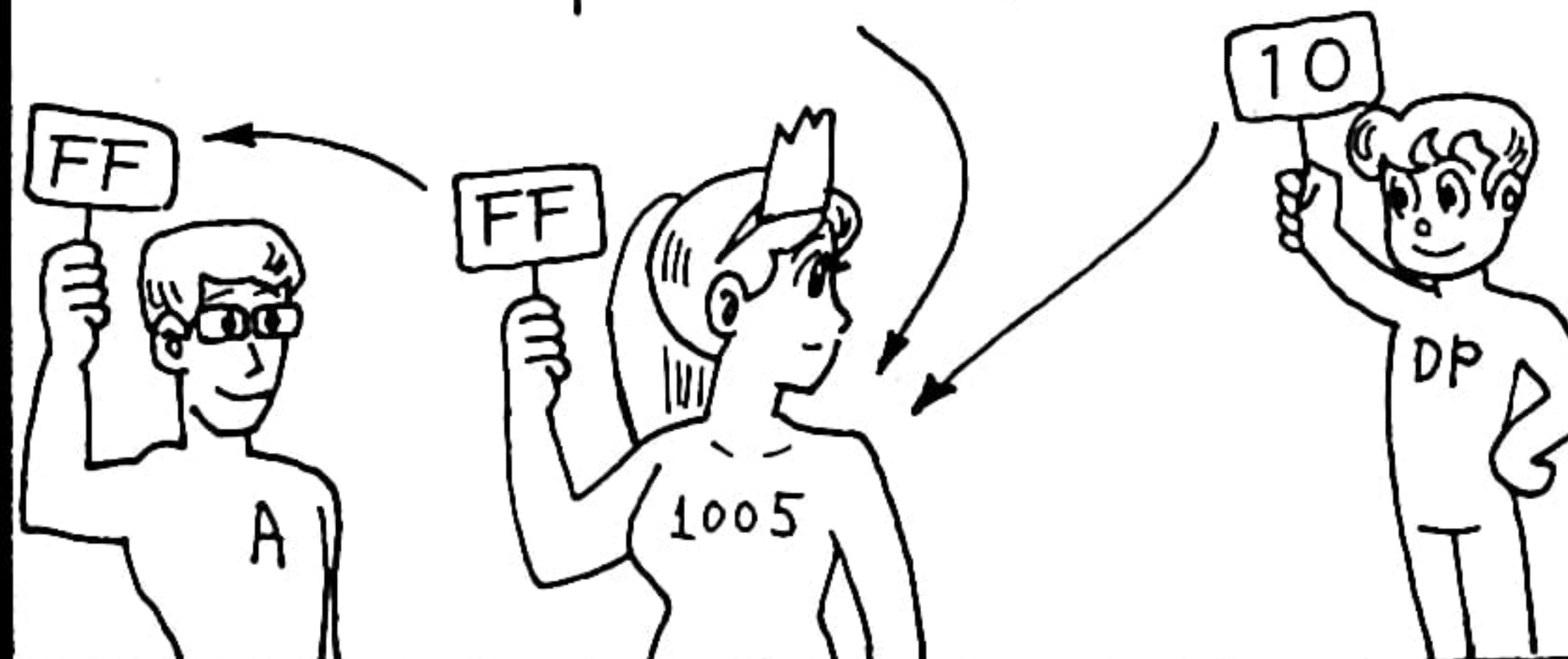
命令一覧表

インデックス		
オペコード	サイクル数	バイト数
A6	4+	2+

?
＋

次はダイレクトモードでニーモニックの識別記号は＄です。

ニーモニック	OPコード	内容
LDA␣＄05	9605	M→A

FF
FF
10
A
1005
DP

アドレッシングモード／オフセットの種類	インデックス インダイレクト	
	ポストバイト（2バイト目のOPコード）	追加バイト
16ビットオフセット	1rr1 1001	2

このrrはX、Y、U、Sの各レジスタです。

オペコードA6のニーモニックはLDAですがオペランドはこれに続くコード（ポストバイト）がないと決まりません。

A6⊕

このあとの2バイトに0120を入れるとA6990120というコードができます。ニーモニックはLDA␣〔＄0120,X〕です。

〔＄0120、X〕

16ビットオフセット

インダイレクト記号

たとえばrrにXを入れると10011001となり99というポストバイトができます。

A6　99

レジスタのコードはこのようになっています。

rr	コード
X	00
Y	01
U	10
S	11

ニーモニック	OPコード	内容
LDA␣〔＄0120,X〕	A6 99 0120	M→A

0120
+2013
2133

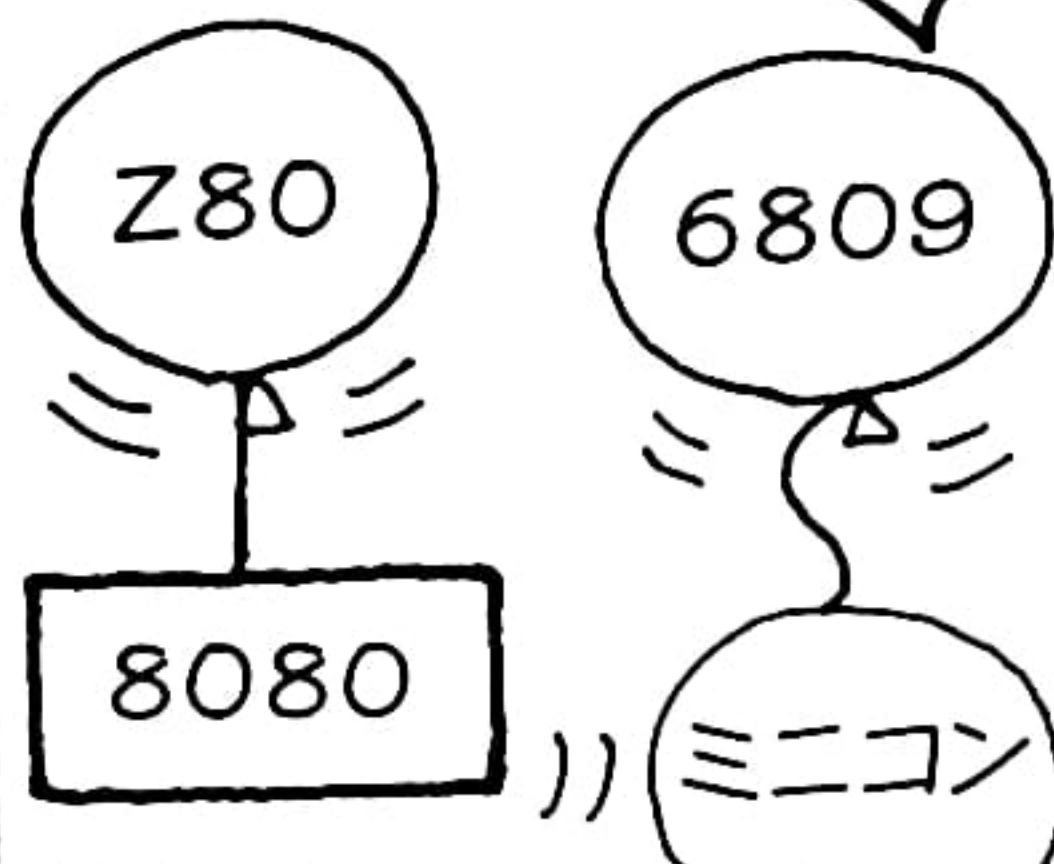
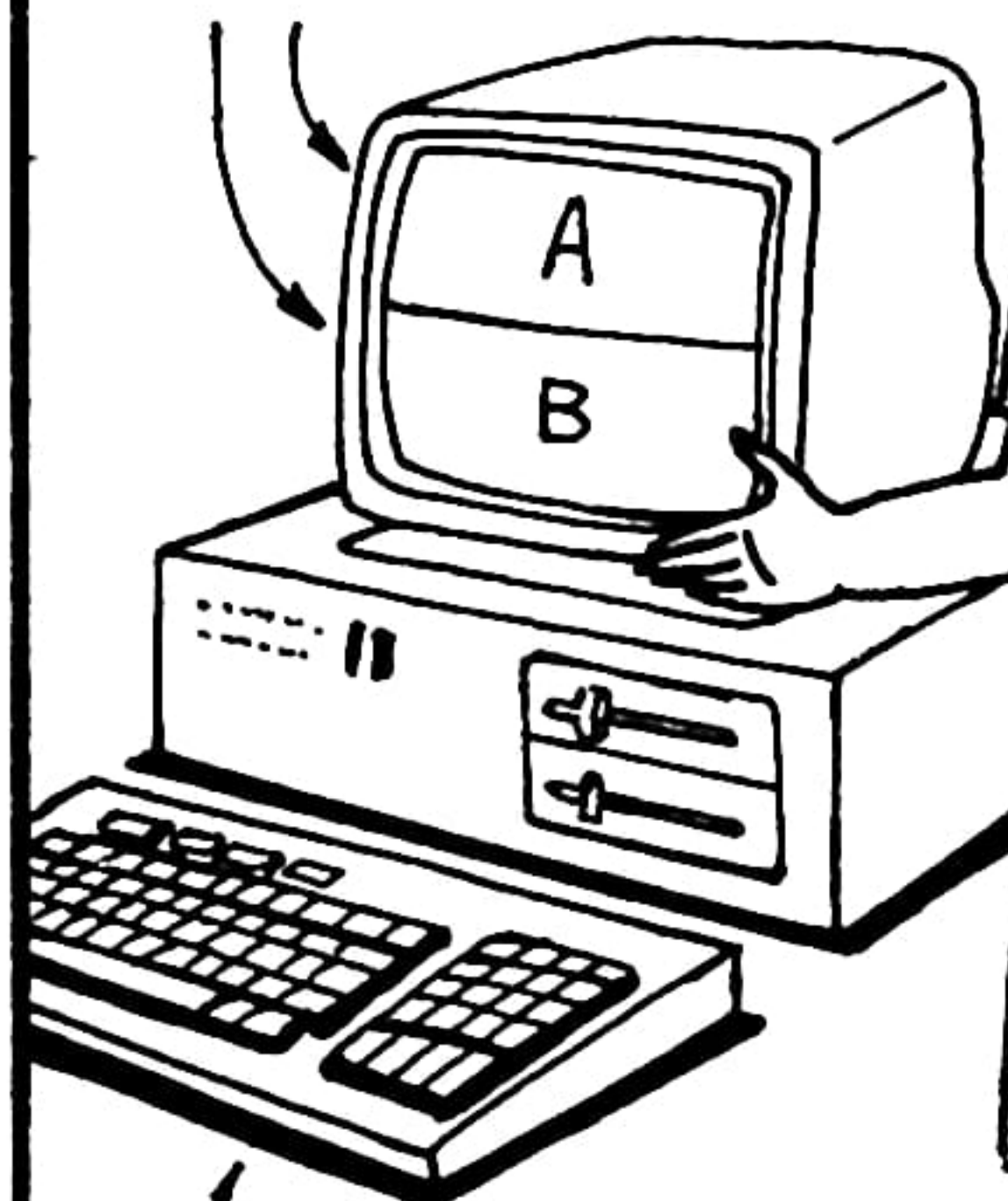
インダイレクト（間接）アドレッシング

こういうのをオフセットといいます。

2013

まどろっこしいね。

28
28
3F
FF
X
00
A
メモリ
3FFF
2133
2134

リラティブモードといい1バイトのものがショート、2バイトのものがロングです。
ニーモニック
OPコード
内容
BEQ␣30FF
1025 30FF
Z＝1なら 30FF＋(PC)にブランチ
30FF
+1003
4102
Z＝1のときのブランチ先
1003
もうひとつジャンプ命令が特徴的です。68系ではブランチ命令と言っているものです。
またFM-11やS1で走るOS-9というUNIXライクなOSもあります。マルチタスクやマルチウィンドーの機能を持っているそうです。
ミニコンOS (UNIX)
パソコンOS (OS-9)
開発ツールとしてはCOMAS-FDTなどがありますが私は体験していないのでここでは触れません。
マシン語
……
こうしてみると6809はZ80にくらべレジスタは少ないのですが、アドレッシングに大きな特徴があるようです。
Z80
6809
8080
ミニコン
現在68系のパソコンでCP/Mを走らせるためにはCPUカードをZ80に換える必要がありますが、そのうち6809用基本ソフトが完成すれば6809の性能をいかんなく発揮することでしょう。
S1
6809の命令は初めのうちはとっつきにくいかもしれませんが慣れると素晴らしいものであることがわかってくるそうです。
マルチウィンドー
別個に表示
A
B
ビィーッ
ビィーッ
マルチタスク
たとえばプリンタが印字中でも別のプログラムを走らせることができる。

（6809命令一覧表の一部）

	インハラント		ダイレクト		エクステンド		イミディエイト		インデックス		リラティブ	
	OPコード	バイト数	OPコード	バイト数	OPコード	バイト数	OPコード	バイト数	OPコード	バイト数	OPコード	バイト数
			96	2	B6	3	86	2	A6	2+		

友子ちゃんの疑問

ニーモニックはLDA。オペランド部はどのように書いたらいいのかしら？

オフセットって何かしら？

インデックスのバイト数2+の+は何かしら？

6809アセンブリ言語　記述例					
	モード	ダイレクト	エクステンド	イミディエイト	インデックス
命　令	オペランド識別記号	$	>	#	オフセット値 レジスタ 〔　,　〕 インダイレクト記号
LDA (M→A)	ニーモニック表記	LDA␣$25	LDA␣>25	LDA␣#10	LDA␣〔$0120,X〕
	マシン語表記	96 25	B6 25	86 10	A6 99 01 20

68系のマシン語に変換しても80系のように上桁下桁が逆転しない。理由は、CPUが上位アドレスを先に読むためです。

マシン語Q＆A

＊マシン語とは？

8ビットCPUでは8ビットで表現されるコードでコンピュータが理解できるON、OFFの信号を表わしたものです。たとえば0011 1110というコードをオペレーションコード読込み（フェッチ）のタイミングでCPUが読み込むと、CPUは次の1バイト（8ビット）のデータをAレジスタに入れるという動作をします。しかし、普通は4ビットずつに区切って16進2桁で表わし、これをマシン語として使っています。0011 1110は3Eと表現します。（16進でプログラムする時はモニタ（プログラム）が必要です。）

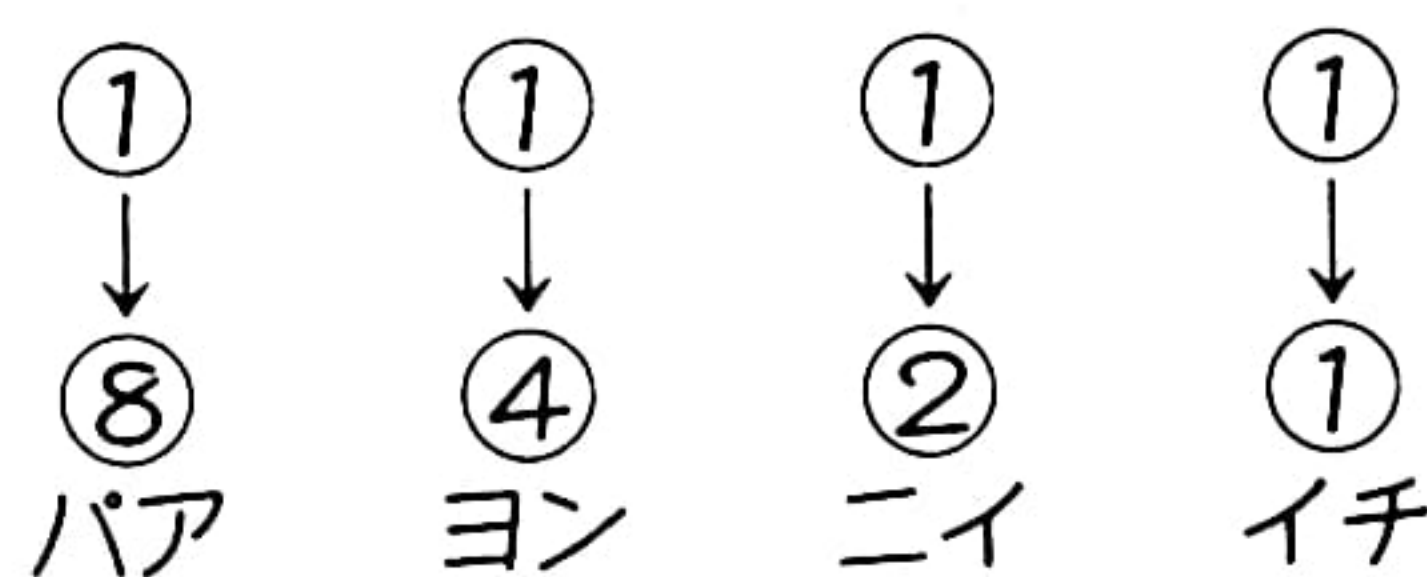
＊2進数と16進数の関連は

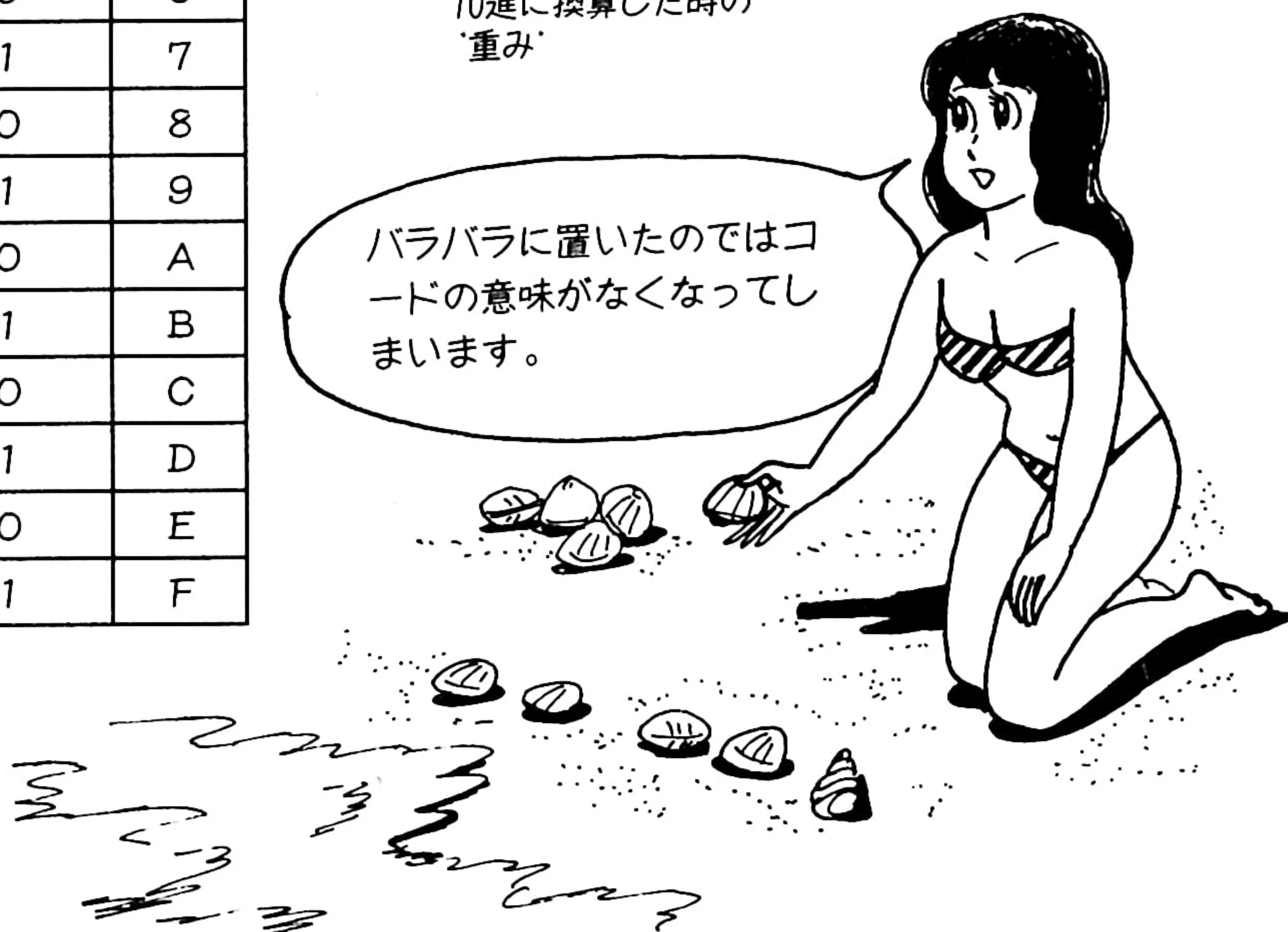
2進数は0か1で表現されますが、これを'ゼロ'と'イチ'と発音すると10進数に慣れている私たちの頭の中は混乱してしまいます。電圧信号と思って'ロー'と'ハイ'と発音するとなんとなくこれまでの数字とは違う世界が見えてくるのではないでしょうか。

10進	2進	16進
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

たし算

$$\begin{array}{r}0\\+0\\\hline 0\end{array}\quad\begin{array}{r}1\\+0\\\hline 0\end{array}\quad\begin{array}{r}1\\+1\\\hline 10\end{array}$$

2進→10進換算のしかた

①	①	①	①
↓	↓	↓	↓
⑧	④	②	①
ハア	ヨン	ニイ	イチ

全部足すと15です。
10進に換算した時の'重み'

＊ニーモニック言語とは

3Eといった16進でプログラムした方が0011 1110という2進よりはわかりやすいのですが、プログラムする時はまだよいとしてもデバッグが大変です。そこで英文字でCPUの動作内容がわかるような形で表現したものが、MVI␣Aというニーモニックなのです。ニーモニックとコードとの間にはCPUを作った会社の変換ルールしかないため、変換表がなくては変換できません。8080、Z80、6809、それぞれニーモニックが違います。このニーモニックコードは、マシン語コードと1対1で対応しているため、これもマシン語という場合もあります。

＊どのCPUのマシン語を覚えるのがよいか

初心者はZ80のニーモニックが使いやすいのではないかと思います。Z80は参考書もたくさん出ているため勉強するにはよい環境です。MSXではCPUとしてZ80相当品を使うことになっています。6809の場合はまだ環境がととのっていないので、よけいな労力を払うことになるかもしれませんが……。

＊どうしたらマシン語を使えるか

ワンボードマイコンの時代にはいやでも16進のマシン語でCPUとトークする必要がありました。ROM型パソコンを持っている人はMONコマンドで16進のマシン語が使えます。(ハンドヘルドやポケコンにはMONモードはありません。) CP/Mの走るパソコンならニーモニック言語が使えます。実用的な使い方としては（BASIC言語＋マシン語）というパターンになります。

＊マシン語は統一されるか

家庭用電源は西日本と東日本では周波数が違いますね。誰も60Hzか50Hzのどちらかに統一しようとは思わないでしょう。影響が大きすぎます。日本では8ビットCPUはZ80を含む80(ハチマル)系と68(ロクハチ)系の2つのマシン語があります。JIS規格で検討されるようになったらあるいはどちらかに統一されるかもしれませんが、あえてその必要もないでしょう。

Ⓒ CPUの中はどうなっているか

ロジック

回路の種類

フリップフロップ

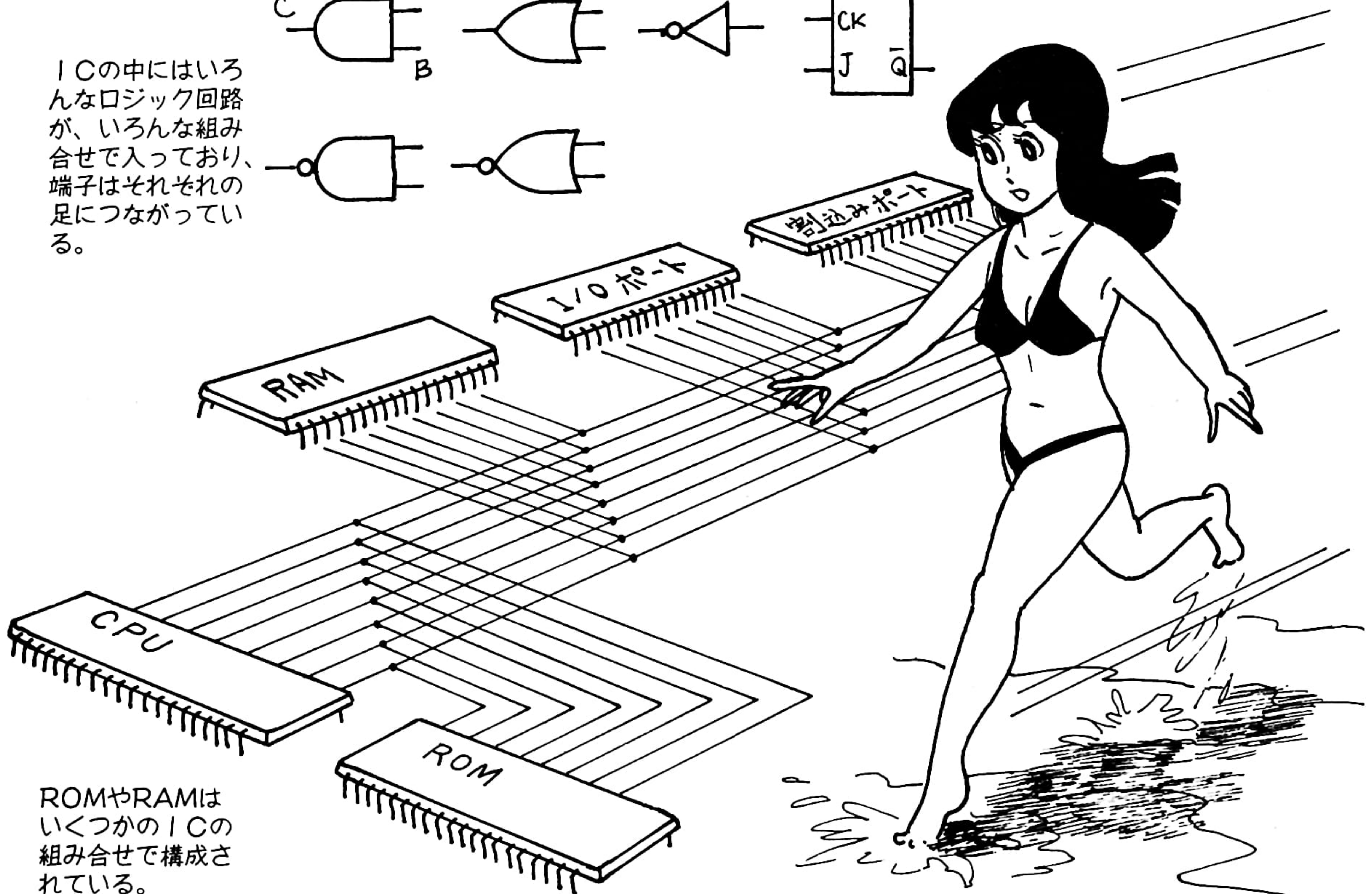
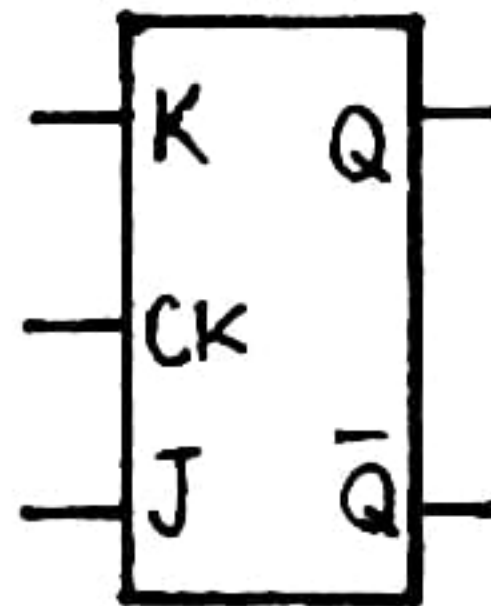
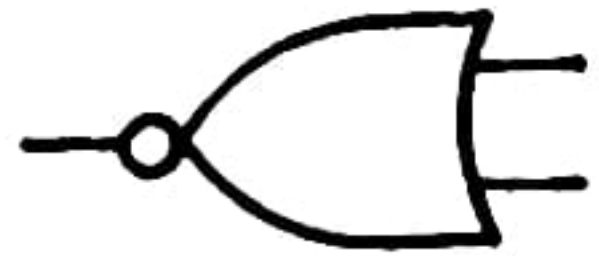
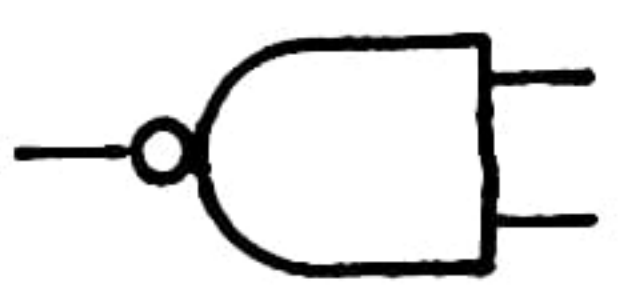
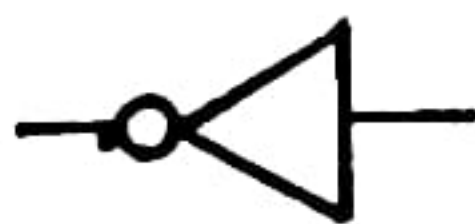
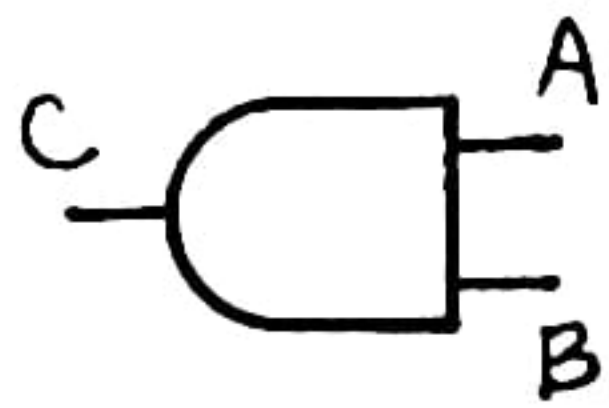
ＩＣの中にはいろんなロジック回路が、いろんな組み合せで入っており、端子はそれぞれの足につながっている。

RAM

I/Oポート

割込みポート

CPU

ROM

ROMやRAMはいくつかのＩＣの組み合せで構成されている。

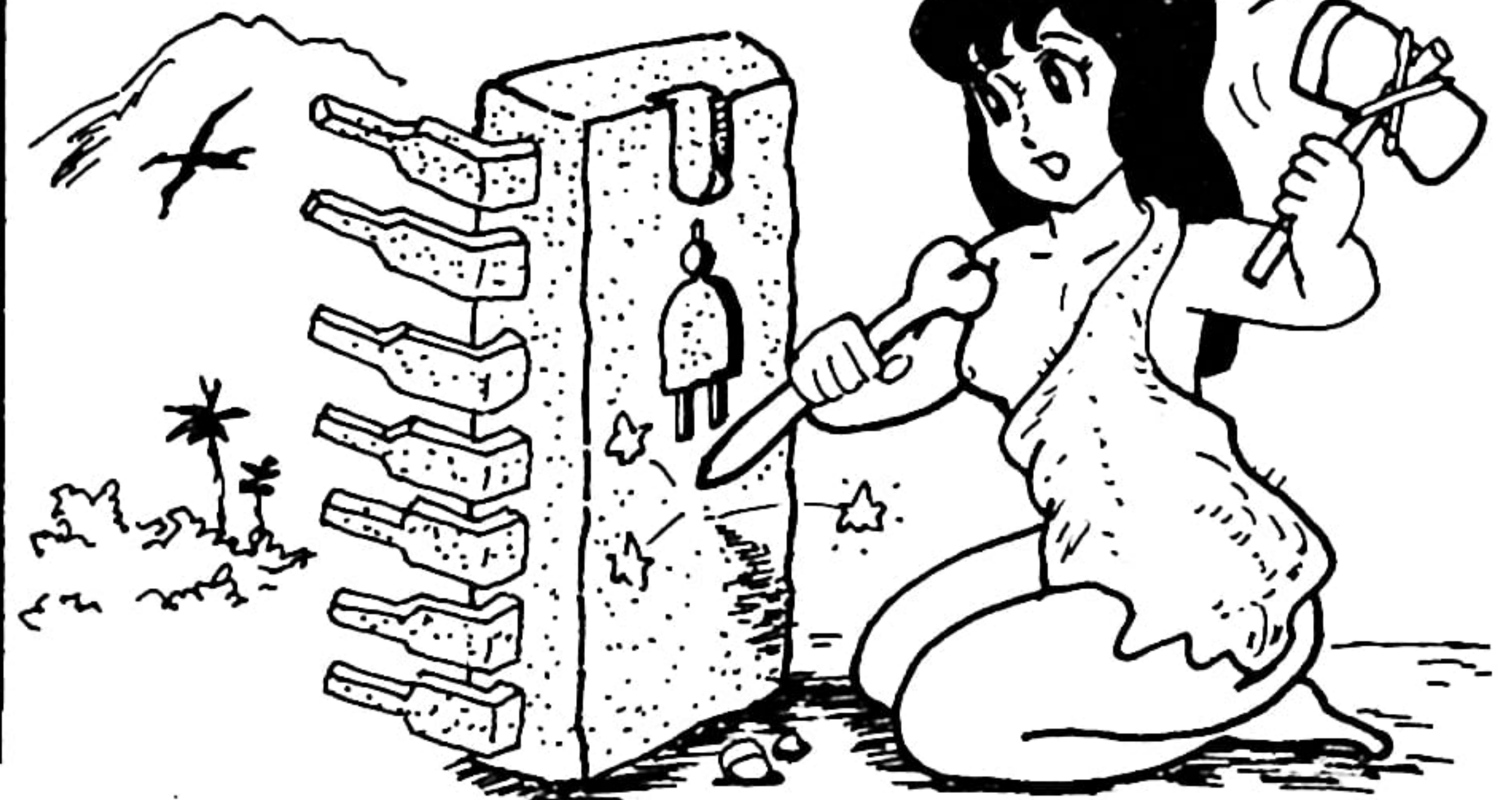
回路のハードの説明は、ちょっとむずかしいので、興味のない人は適当にとばして読んでください。

①石(IC)の大きさ

アイシー

ICの技術は日進月歩もいいとこ、とても目まぐるしく改廃が行なわれています。価格も、どんどん変わっていきますよ。

SSS
Z-80 2,800
8085 4,000
8080 1,800
8224 400
8228 1,200

ザイログ
モトローラ
フェアチャイルド
テキサスインスツルメンツ
ロックウェル
インテル
NEC
東芝
三菱
日立

ちょっとむずかしいのが、フリップ・フロップ。ICの集積度も比較的高くなっています。

IC

うーん、ちょっと(頭が)重いわ

14、16ピン

一番簡単な論理素子がゲートと呼ばれるものです。ICの集積度も少ないものです。

14ピン

だから日ごろからどんなICがあるか見慣れておかなければ、必要な時にすぐに回路を組むことができません。

－74だ

D Q
CK $\bar{Q}$

が、ほしい。

IC回路図に、-368、-74、-174、-LS368、-08、-04とか書いてあるのは、すべてTI社のもので前にSN74が略されているのです。

SN74368
SN7474
SN74174
SN74LS368
SN7408
SN7404

ゲートやフリップ・フロップなんかはテキサス・インスツルメンツ社の製品番号で示すのが一般的です。たとえばANDゲートなら-00、これはSN7400のことを指しています。

テキサス インスツルメンツ社
またの名を
TI社

ところがマイコンのCPUクラスになると、とても、とても。ハハハ…(と、笑ってゴマかすの)

40ピン

CPUのハード的なお話をする前に回路を構成するのに必要なICについての予備知識を説明しましょう。

IC

こんな小さなパッケージの中にどんどん情報が詰め込まれていきつつあります。これに何を入れたら何が出てくるのかをしらねばなりません。

10101011
11101100
10100100
11100100
11011011
00011011
00110011
00101100
01101100
01010000

また、先頭に8のついている、たとえば8080、8216、8224などはインテル社のものです。ICの場合、この番号が社名や仕様を表わす大事な意味を持っています。もちろん、これは登録商標にもなっています。

8080
8212
8251
8255
8259
8257

②ロジック用IC

うまく考えられたシンボルマークです。

ロジックにはAND(アンド)、OR(オア)、NOT(ノット)などがあり、ロジック回路を構成するうえで不可欠のものです。ANDやORには最低2入力が必要です。

AND

OR

NOT

たとえばANDの場合を考えてみます。AとBの2入力の場合(2ビットと考える)、入力の状態は4通りとれます。それぞれの状態に対する出力Cの状態がANDの場合はこうなるのです。

真理値表

インプット		アウトプット
A	B	C
0	0	0
0	1	0
1	0	0
1	1	0

出力　入力

この表の1や0はそれぞれ電圧信号の有(5V)、無(0V)に対応しています。それで1はHIGH(ハイ)、0はLOW(ロー)と呼んでいます。

+5V　ハイ　1

ロー　0V　0

また、ロジックを表わすシンボルで一番多く使われているのはアメリカのMIL規格のものです。

MILITALY

出力側に小さなマルがつくと否定を表わします。たとえばANDの先にマルのあるものはNOT・ANDとなりNAND(ナンド)と呼ばれます。ORの場合は同じやり方でNOR(ノア)となります。

NAND　NOT AND

NOR　NOT OR

これはNANDゲートの動作を表わす表です。むずかしくいうと真理値表です。

0　1　1

インプット		アウトプット
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

シンボル化

ゲートのおしりにも意味があって、まっすぐはAND、弧になっているものはORを示します。

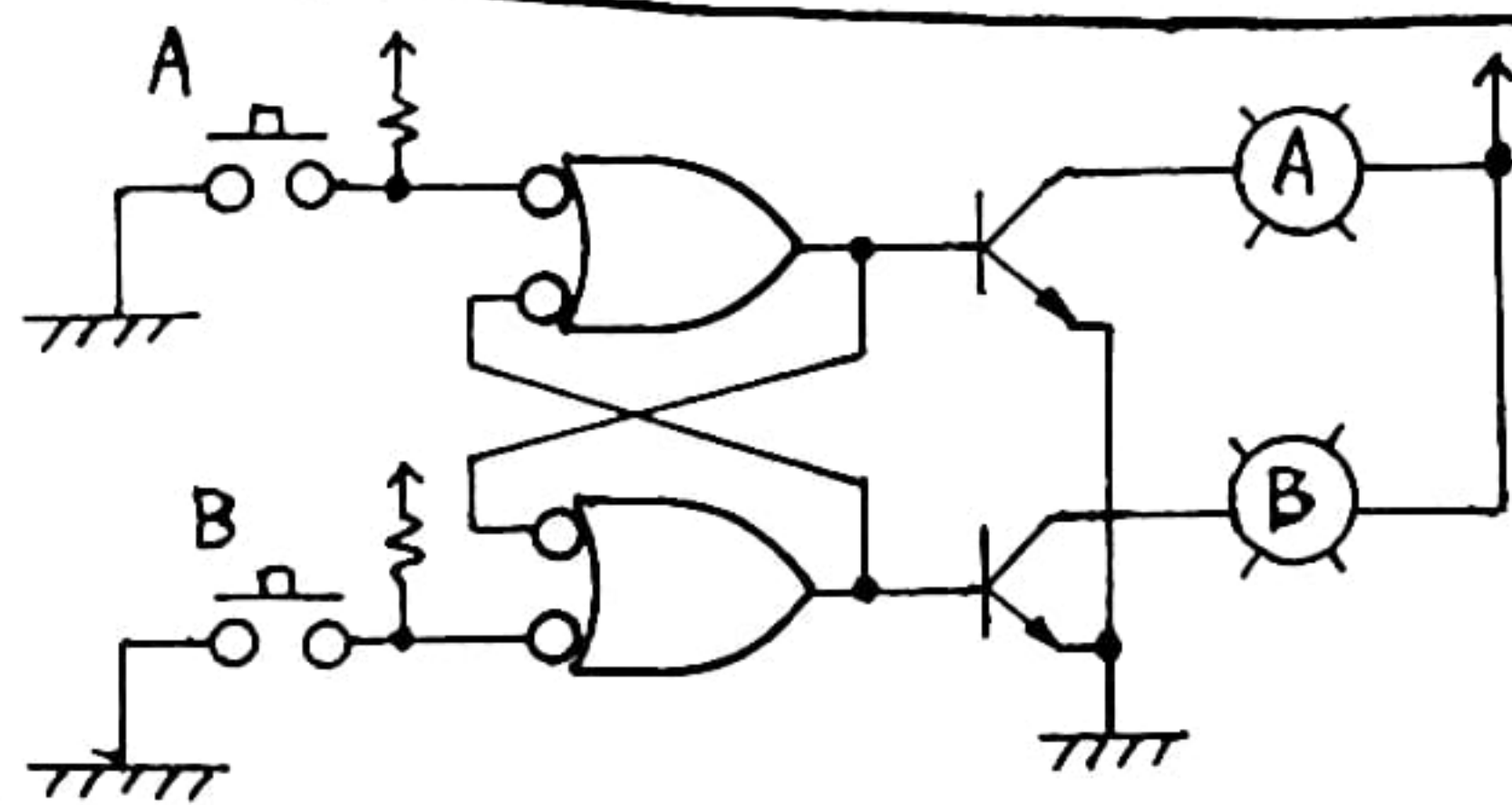
ゲートのおしり

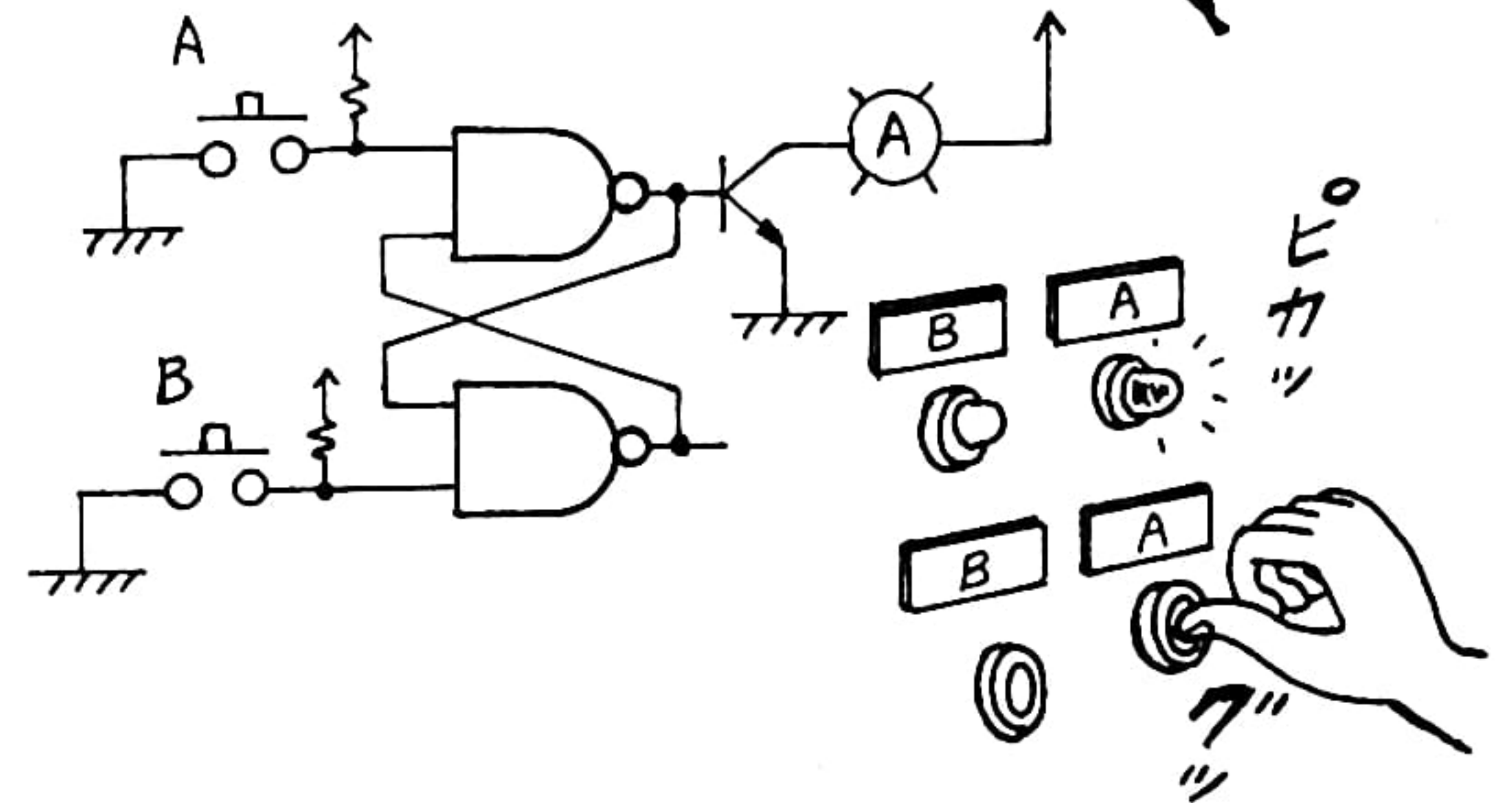
またNANDゲートでも出力が0の時の条件を強調しない時と1の時を強調したい時ではシンボルが変わります。

入力がABとも1で出力が0となる。

入力がABいずれか0のとき出力が1となる

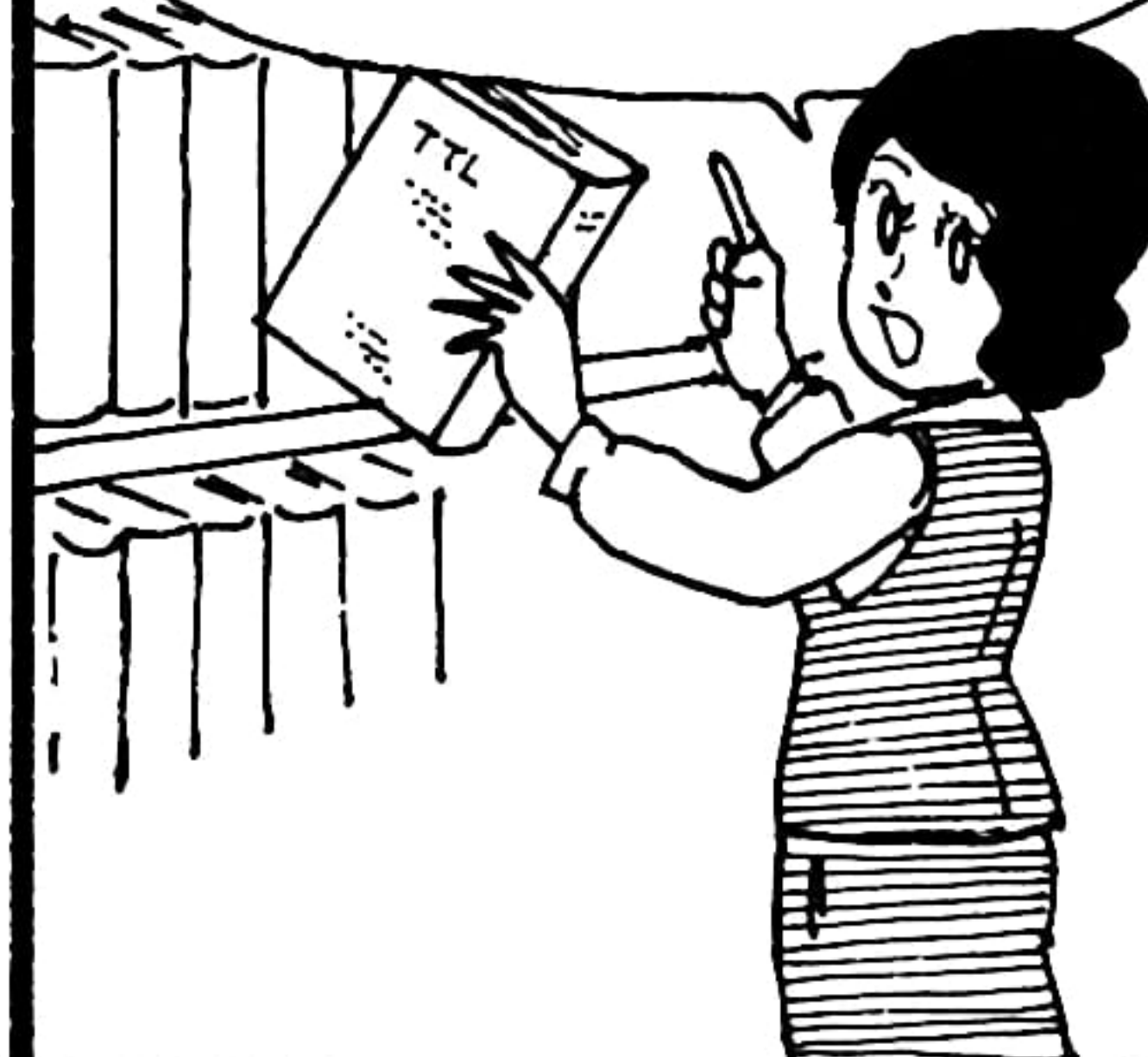
ところがこの回路はスイッチを押した時、つまり入力が'0'になった時、出力が'1'になってその状態が維持されますのでこう表現した方がわかりやすくなります。

A B

これはNANDゲート2つを使ってできるフリップ・フロップという電子的な自己保持回路です。

A B ピカッ グッ

テキサスインスツルメンツ社のTTLロジックICは応用製品のディジタル化と共に一世を風靡した感がありましたね。

アナログ指示計

ディジタル指示計

ICの初期のころは、この比較的小さな回路のゲートから製品化されたんです。

これらのICはゲートとも呼ばれています。

アウトプット

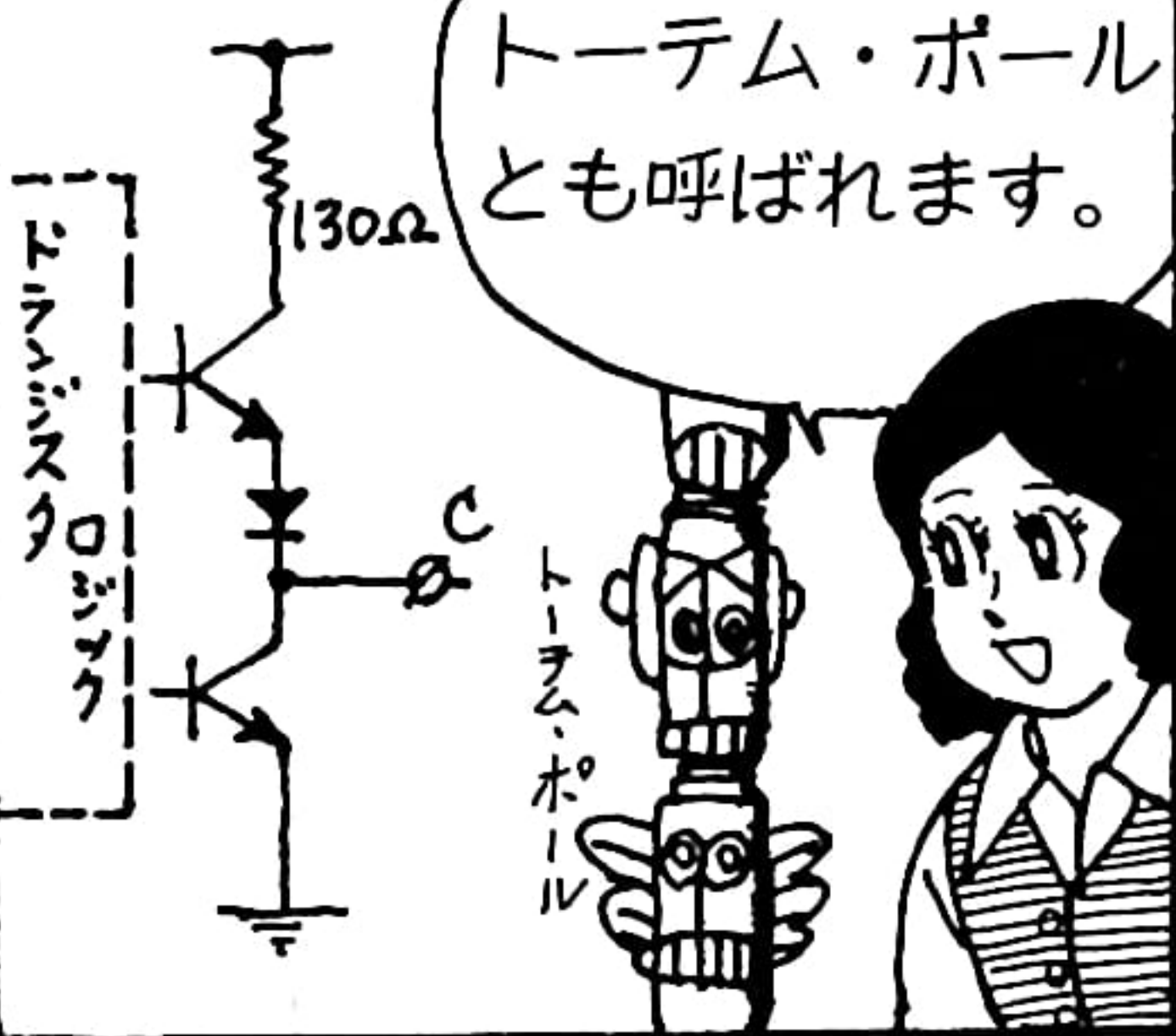
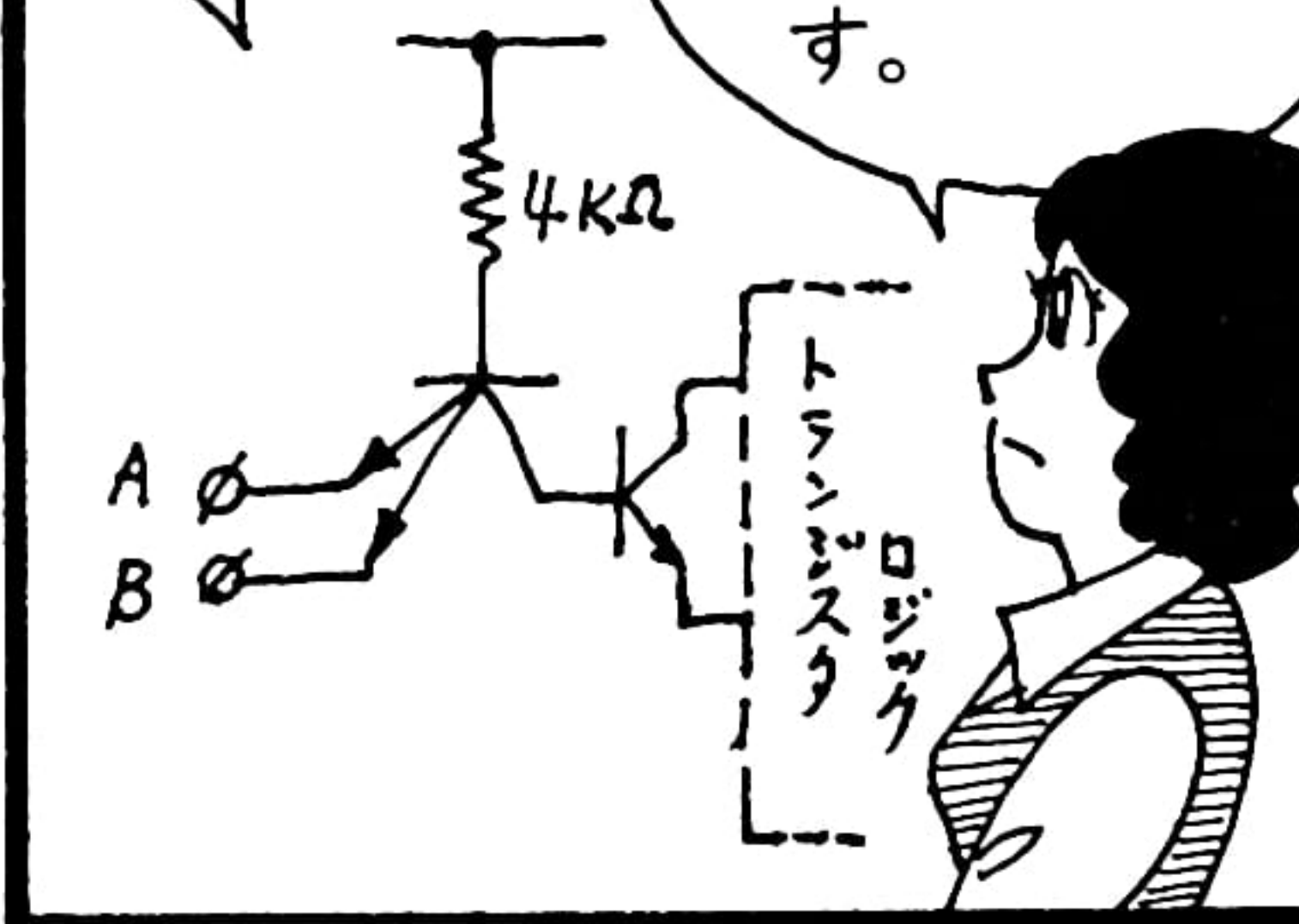
GATE

インプット

これはゲートの出力回路の部分でトーテム・ポールとも呼ばれます。

130Ω

トランジスタロジック

トーテム・ポール

これはゲートの入力回路です。

文字通りTTLはトランジスタ回路で論理を構成しています。

4KΩ

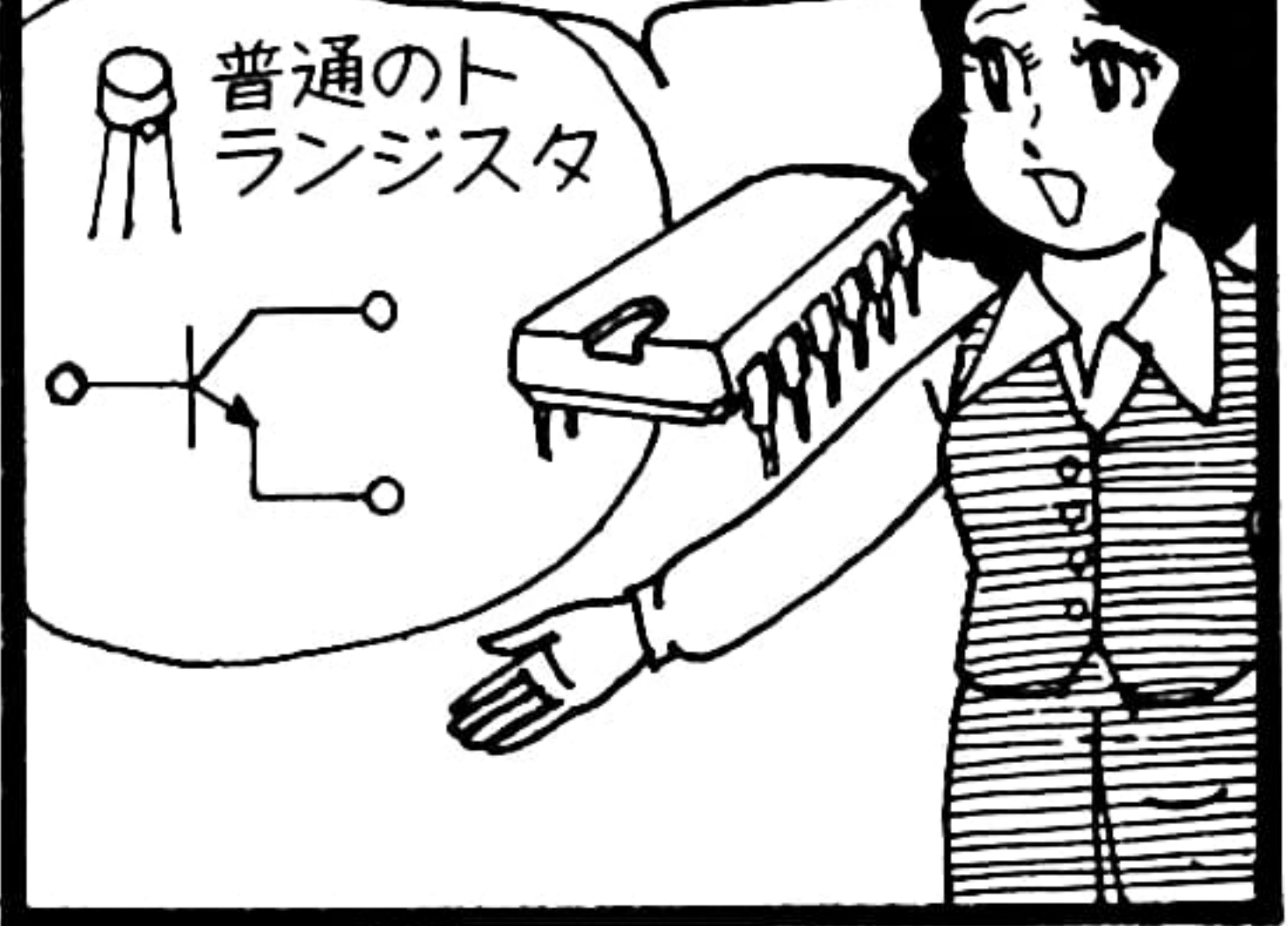
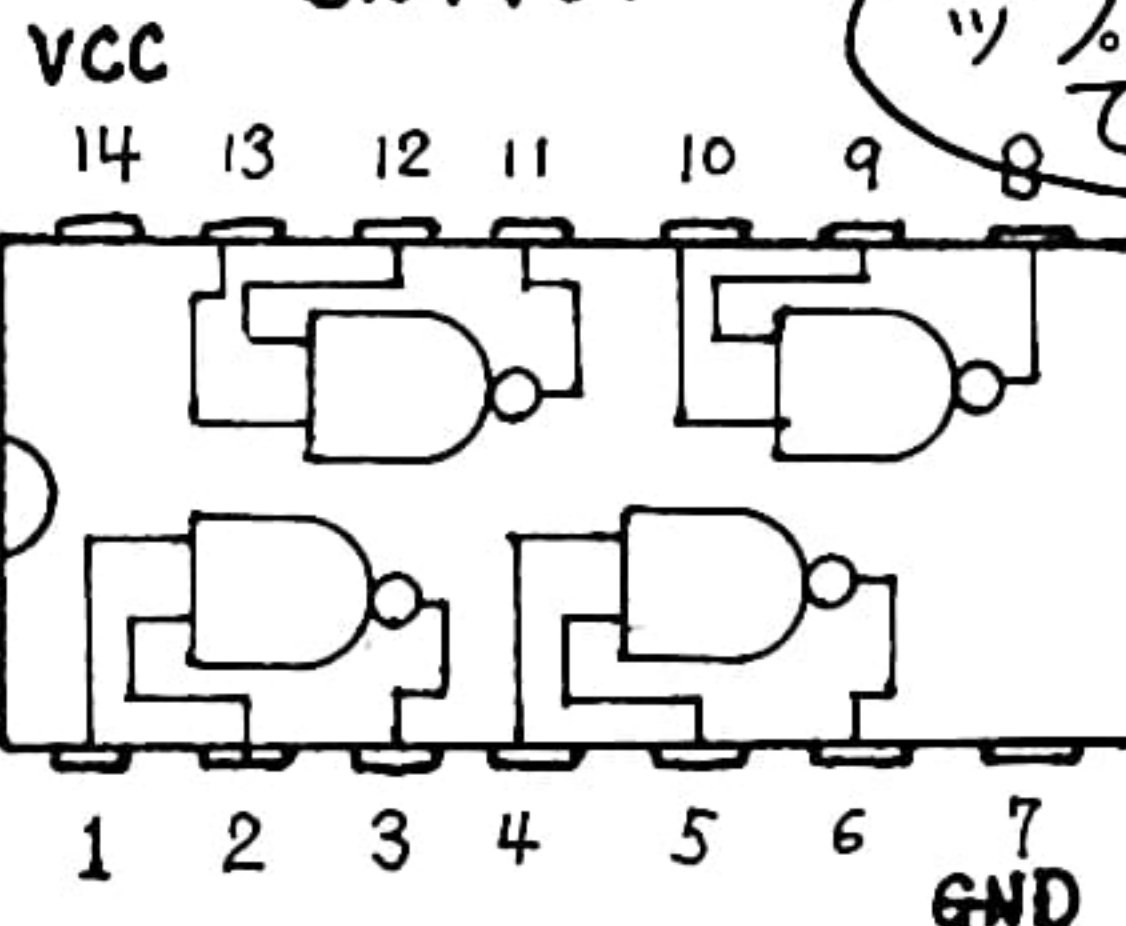
トランジスタロジック

TTL(ティティエル)というのはトランジスタ・トランジスタ・ロジックの略です。

普通のトランジスタ

テキサスのIC SN7400

VCC GND

ゲートは何個かがひとつのパッケージ(チップ)に入っています。

カタログなどに載っているこのマークはICのくぼみでピンの番号を知る目印です。

ここで決まる

NAND AND NOR OR

VCC(電源 +5V)

4KΩ 130Ω

ロジック

GND

VCC

つまりロジックの部分の組み合せでいろんなゲートができるんです。

ICを使ってディジタル回路の設計をしている時は、もうゲートの中身まで考えなくてもいいのです。入・出力の条件だけは、マニュアルを見て頭に入れておく必要はありますけどね。

いちいちトランジスタ回路で論理を考えていたら大変、プログラムを２進表現で考えているようなものです。

ICの回路図はこのようなシンボルの組合せです。

H 1 2 H 3 L L 4 5 H 6 H

Vcc 抵抗大 I 1 2 3 I 1.6mA 抵抗小 Sink状態(LOW) GND

Vcc 4 5 抵抗小 6 抵抗大 カット・オフ状態 GND

FUN-OUT

ファンアウト

I

これで、何個のゲートがつながるかもわかります。

OUT・PUT

Hの時は、電流が流れ出る。

出力のH・Lの条件

Lつまり、Sink状態では、電流が流れ込んでくる。

許容電流は約1.6mA

IN・PUT

Hの時にゲートに流れ込む電流

約1.6mA

Lの時にゲートから流れ出る電流、これがFUN-OUT 1に相当する。

入力の条件と、

現在はTTLに代わってC・MOSのICが主流になりつつあります。応答はTTLに較べると遅いのですが消費電力がぐんと少ないのが魅力です。

シーモス IN OUT

IN 遅れ OUT Time

消費された電力は熱となります。だからTTLではあまり高密度の回路は望めないのです。

TTLのICは焼損するというトラブルが多い。

TTL回路の問題は、その性質上、常に電流が流れているから消費する電力が多いということです。

TTL

DC5V

I

ATR

ICを取つけ配線していくことを実装するといいます。

A B C

1 2 3

どんな回路でも組めるようになっているのがユニバーサル基板です。

プリント配線

エポキシ樹脂基板

これはICを配線するための板でプリント基板というんです。

最初の一枚目にはお金がかかることになりますけれど。
そこで何回も使う回路は、手配線の部分をプリント配線にしてしまうんです。
パターン設計費
ネガフィルム費
手配線（ユニバーサル基板）の欠点
配線ミス
ハンダ不良
時間がかかる。
個人差がある。
いちいち人が配線するのでは、信頼性や能率から見て量産には向いていません。
ユニバーサル基板では、ごらんの通り配線だらけになってしまいます。
手配線という
プリント配線と、部品の実装は反対側になっています。
パターン面
IC
実装面
できあがったプリント基板にはICや、抵抗・コンデンサをはめこむだけでいいんです。
実装面
7400
プリント線
パターンの設計とフィルムのネガが必要となります。
これは、写真と腐食の技術です。
回路図
パターン図 表と裏の2マイ
プリントキバン
ラックまたはシャーシ
コネクター
コネクターからの配線が、まだゴチャゴチャしています。
プリント基板は、ラックの中にひとつにまとめます。
番地といっても、マイコンのものとは、全然違います。図面の中のICが基板のどこに実装されているのかが、わかるようにするためのものです。
このゲートは基板のA-1番地についている。
基板にこんな記号がついていますが、これは、ICの番地を示すためのものです。
このICは B-3番地についている。
エイッ
パソコンにもCPUやメモリ以外にたくさんのICが使われています。
パカッ
ICは、プリント基板の上にとりつけられています。
標準化された製品でないと、高くついてしまいます。
これも、プリント配線にしてスッキリとさせたのがマザー・ボード。
ミニファンをつけて ICを冷却させる。

③フリップフロップ

ギッタン バッタン

Q

$\overline{Q}$

FF

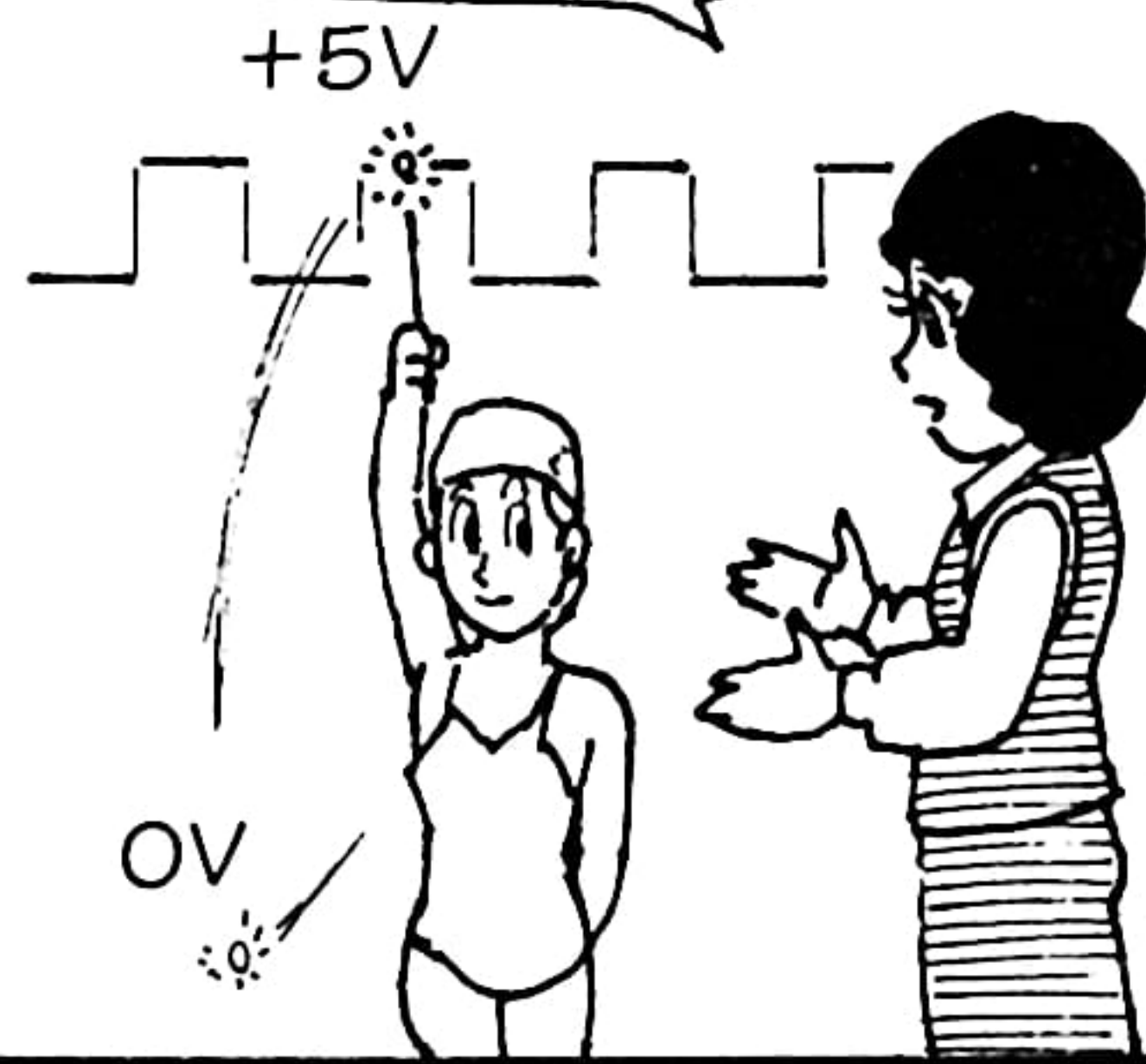
フリップフロップ回路には情報をためておける。

A B Q $\overline{Q}$

こんな形をしたFFはICパッケージの製品としては発売されていません。

必要ならば NANDゲート2個で作ります。これでこのFFが実現されたことになります。

A B Q $\overline{Q}$

Vcc 14 13 12 1 2 3

パッケージに入っているFFで代表的なものはエッジトリガー型JKとD型の2種類です。

J K Q CK $\overline{Q}$

D Q CK $\overline{Q}$

この2つはクロックと同期をとるように考えられたものです。

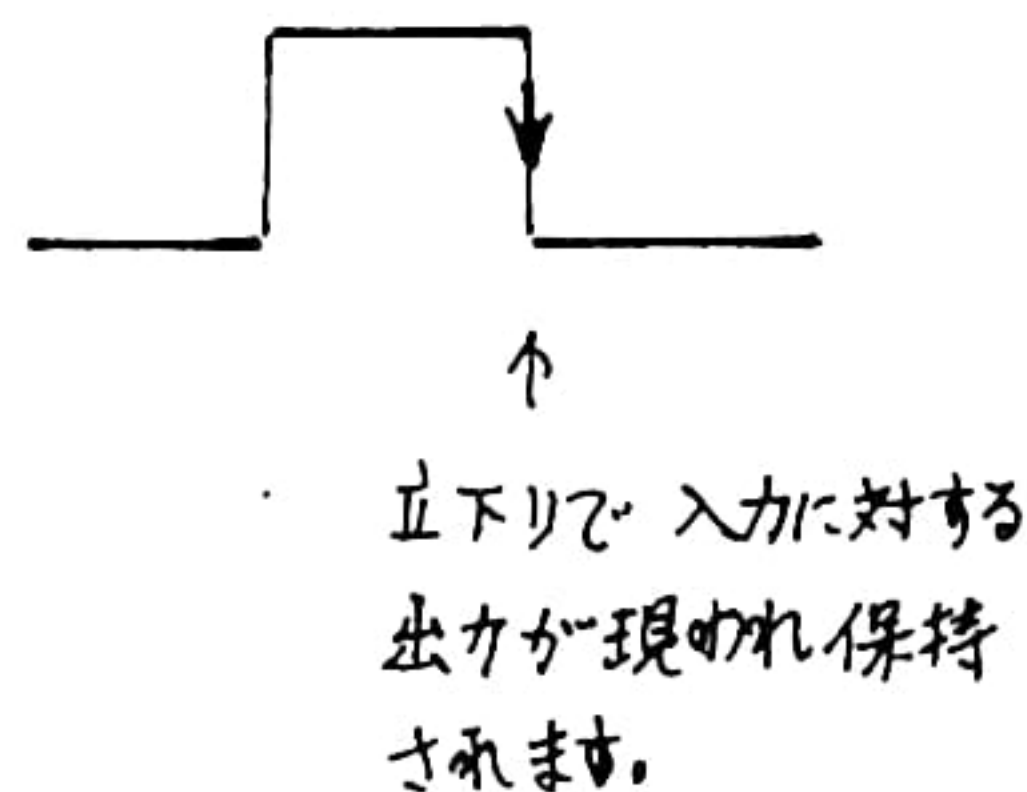
+5V

0V

クロックパルスの立上り、または、立下りのエッジによって、出力が変化するものです。

入力 H

クロック

出力

この、入力に対して出力を変化させる引き金になるものをトリガーといいます。

ズドーン

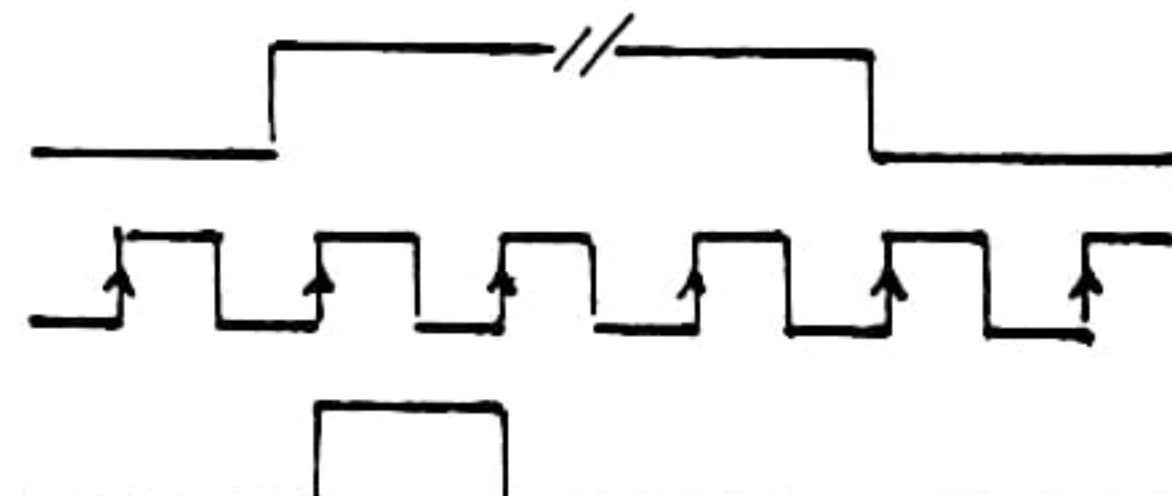
パルス

トリガ

JKタイプのFFは負エッジのトリガーを持ち、

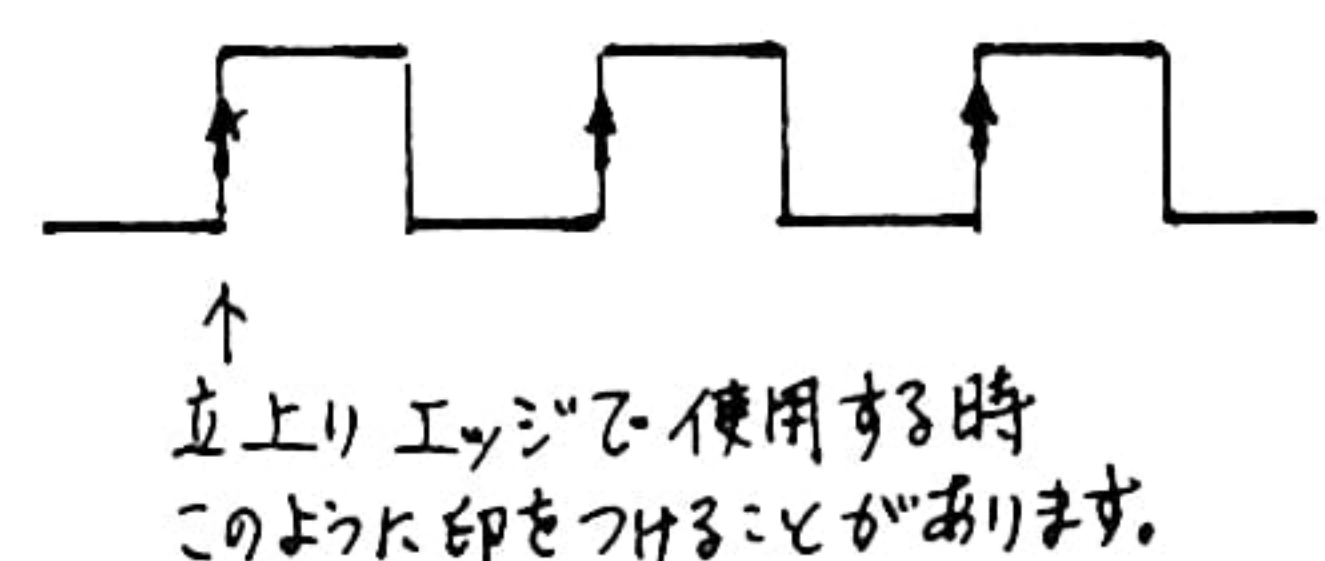
立下りで 入力に対する出力が現われ保持されます。

DタイプのFFは正エッジのトリガーを持っています。

立上りで 入力に対する出力が現われ保持されます。

これらのエッジトリガー方式のFFはクロック1周期に1回しか入力値を出力する機会がありませんから、かえって応用がいろいろ効きます。またノイズにも強いのです。

立上りエッジで使用する時このような印をつけることがあります。

たとえばDタイプのFFを使って不確定長のパルスをクロック1周期長のパルスにけずることができます。

また、そのさい入力パルスの最初または終わりのどちらかで出すことも可能です。(もちろん、回路は違います)

パッケージにはふつう２個のＦＦが入っています。どのピンがどの信号かはこのようなトップ ビューがカタログに出ているのでわかります。

VCC 14 13 12 11 10 9 8

CK Q D Q̄ D Q̄ CK Q

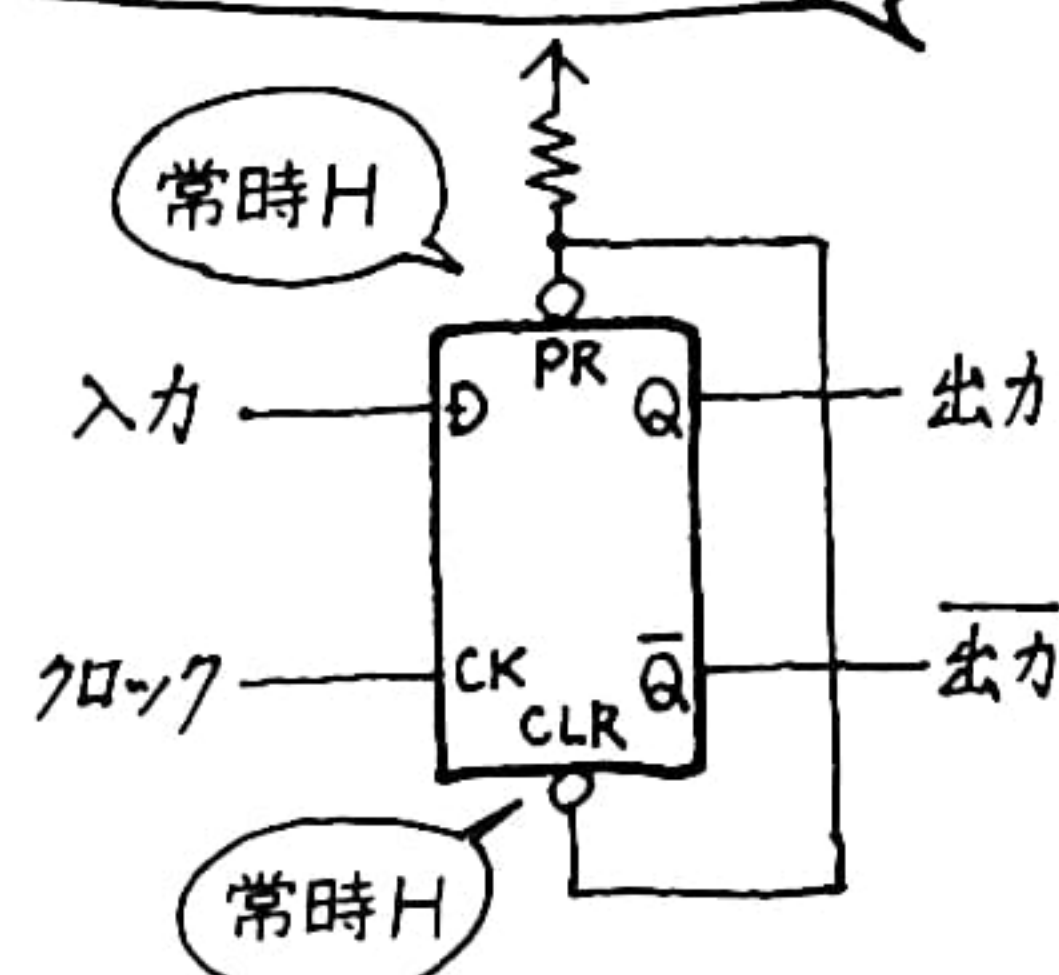
1 2 3 4 5 6 7 GND

ＤタイプＦＦは回路図には、こんな形で通常書かれます。プリセットとリセットはプルアップ抵抗を介して常時Ｈにします。

常時Ｈ

入力 D PR Q 出力

クロック CK CLR Q̄ 出力

常時Ｈ

ＤタイプＦＦはこのような動作をします。

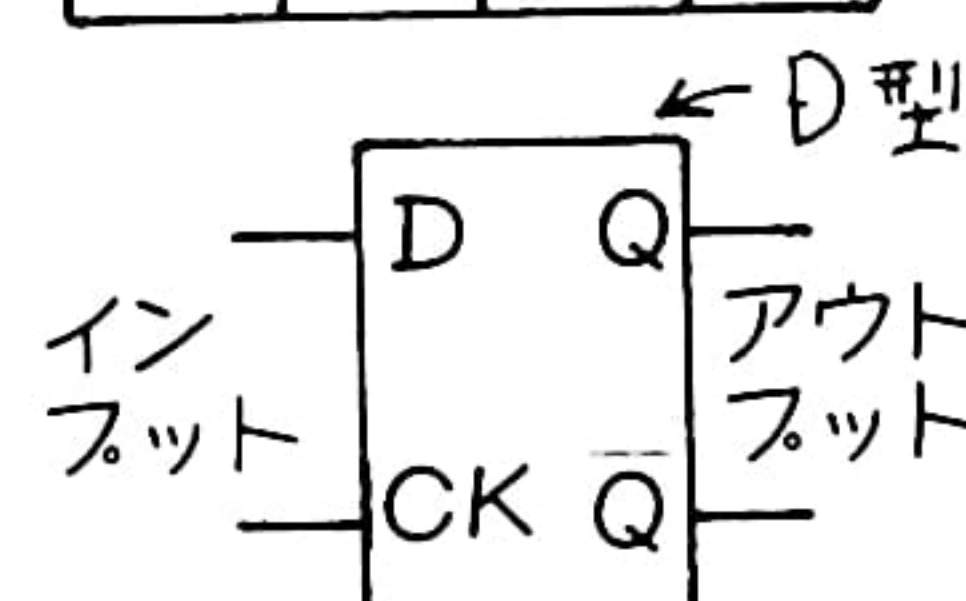
インプット		アウトプット	
CK	D	Q	Q̄
↑	H	H	L
↑	L	L	H

A型

D型

インプット D Q CK Q̄ アウトプット

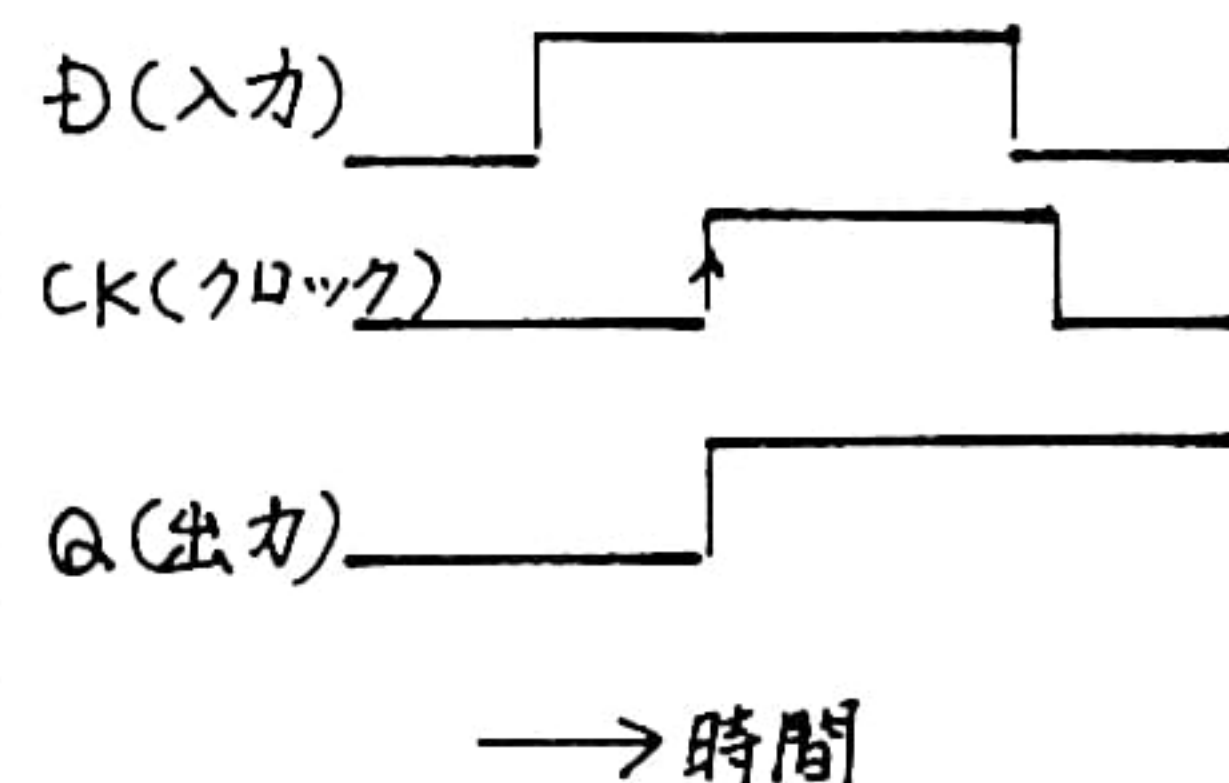
１発目のクロックで出力ＱがＨにセットされ保持されます。２発目のクロックで入力がＬだったら出力ＱはＬに落ちます。

H L D CK Q

まず、正のエッジトリガーと、入出力の関係を示すためにタイミングの例を示します。

D(入力)

CK(クロック)

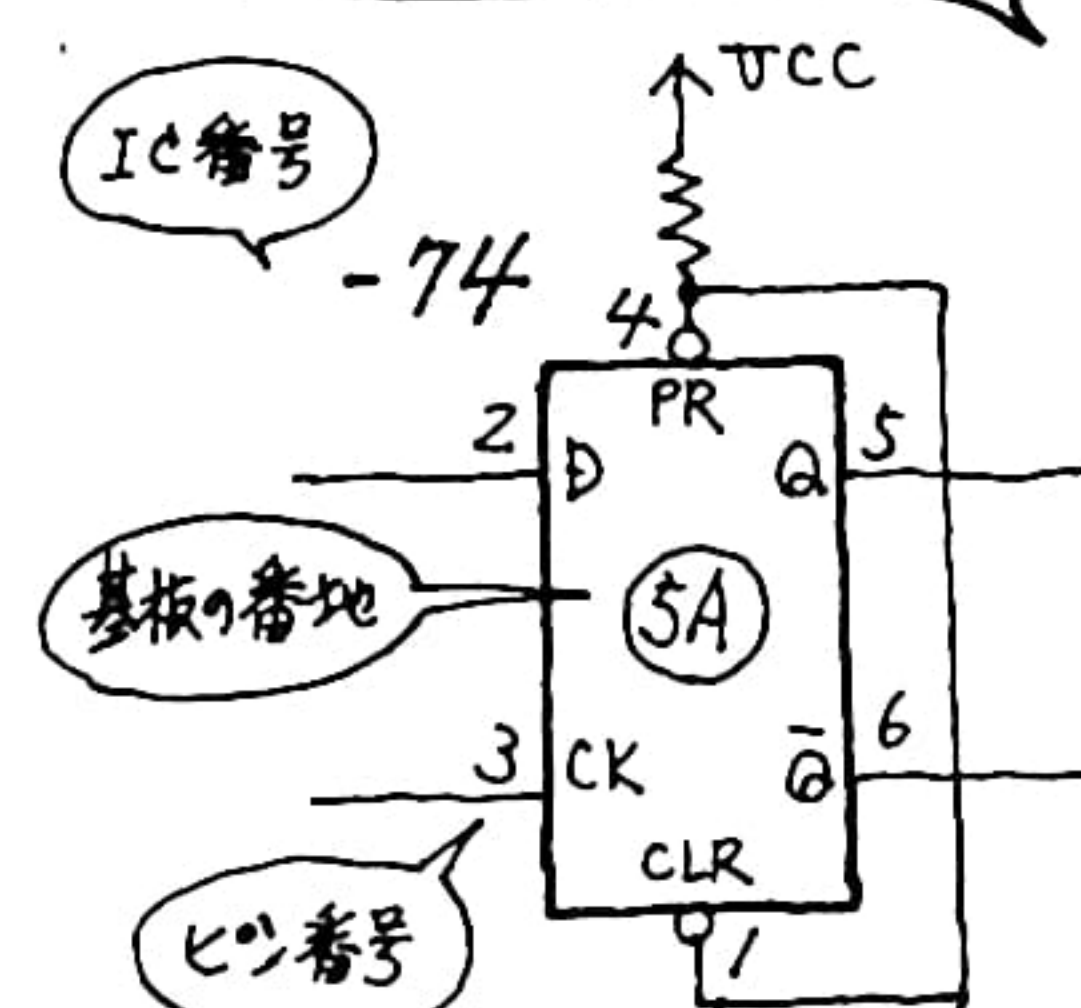
Q(出力)

→時間

図面として書く時は、パッケージNo.、記号、ピンNo.、基板の番地を書いておきます。

IC番号 -74

VCC

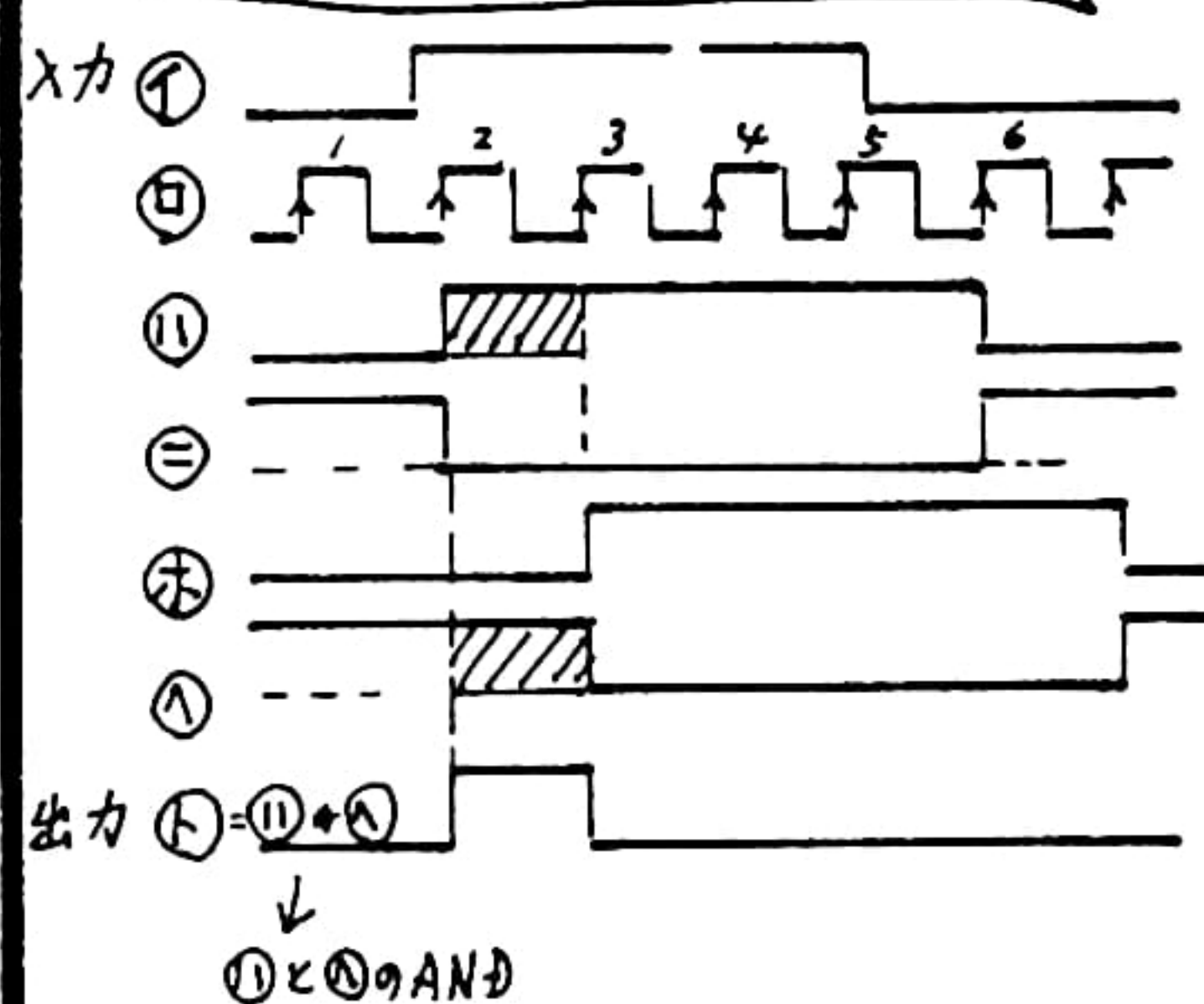
基板の番地 5A

ピン番号

タイミングチャートを書いて回路が合っているか確かめてみましょう。

入力 ㋑ ㋺ ㋩ ㋥ ㋭ ㋬

出力 ㋣＝㋩＋㋬

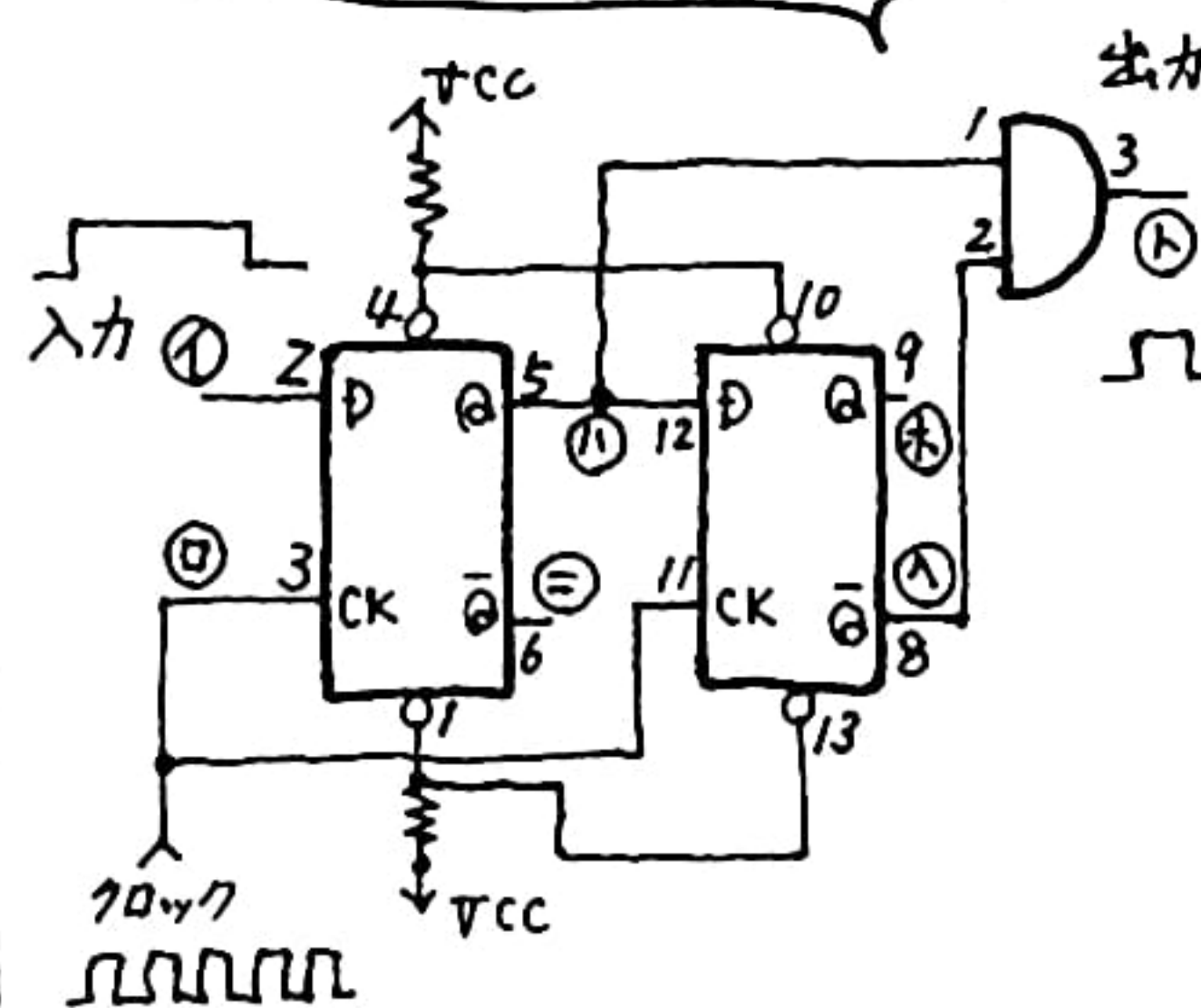
↓

㋩と㋬のAND

この性質を利用してクロック１周期分のパルスをつくることができます。これがその回路です。

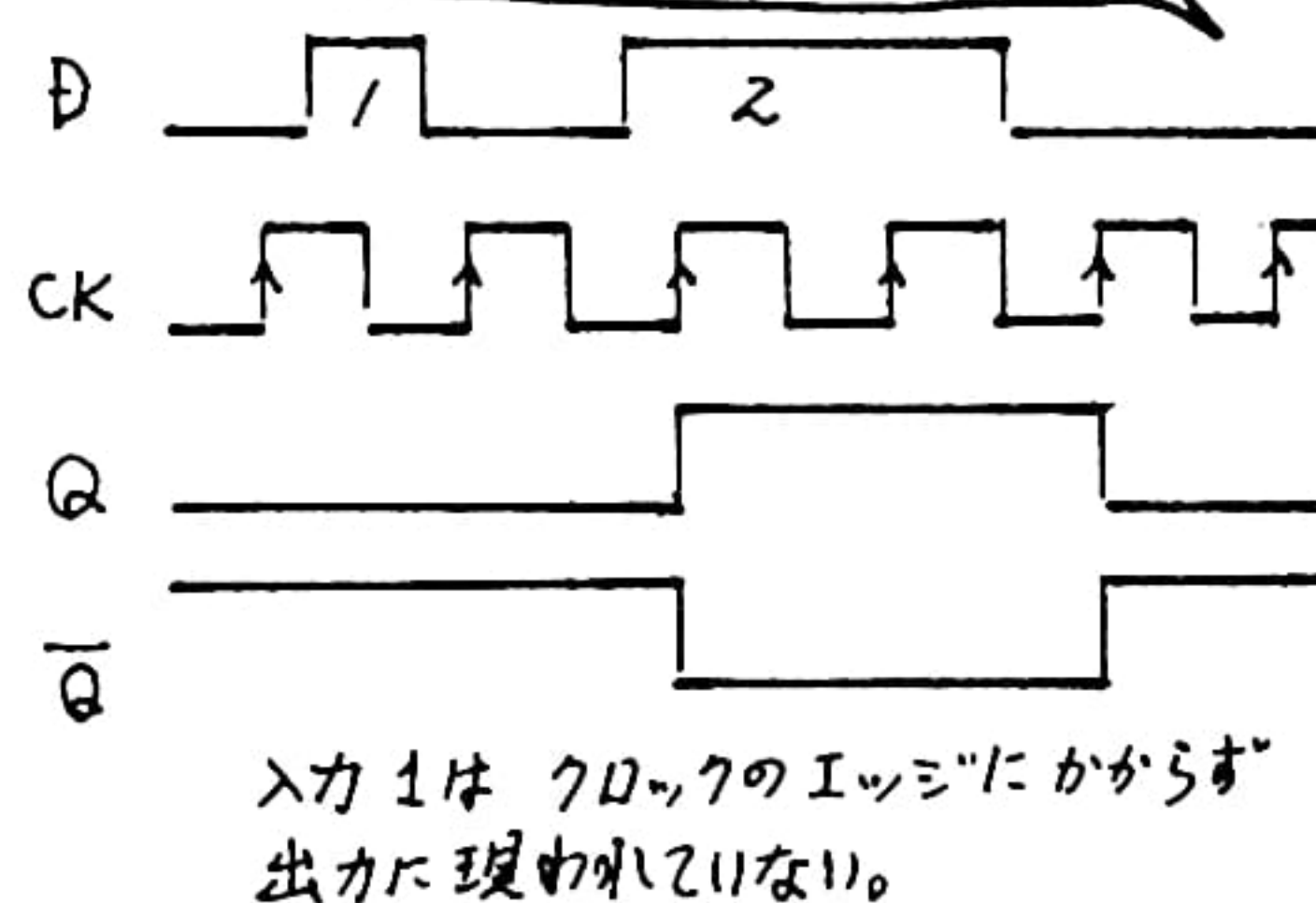
VCC 出力

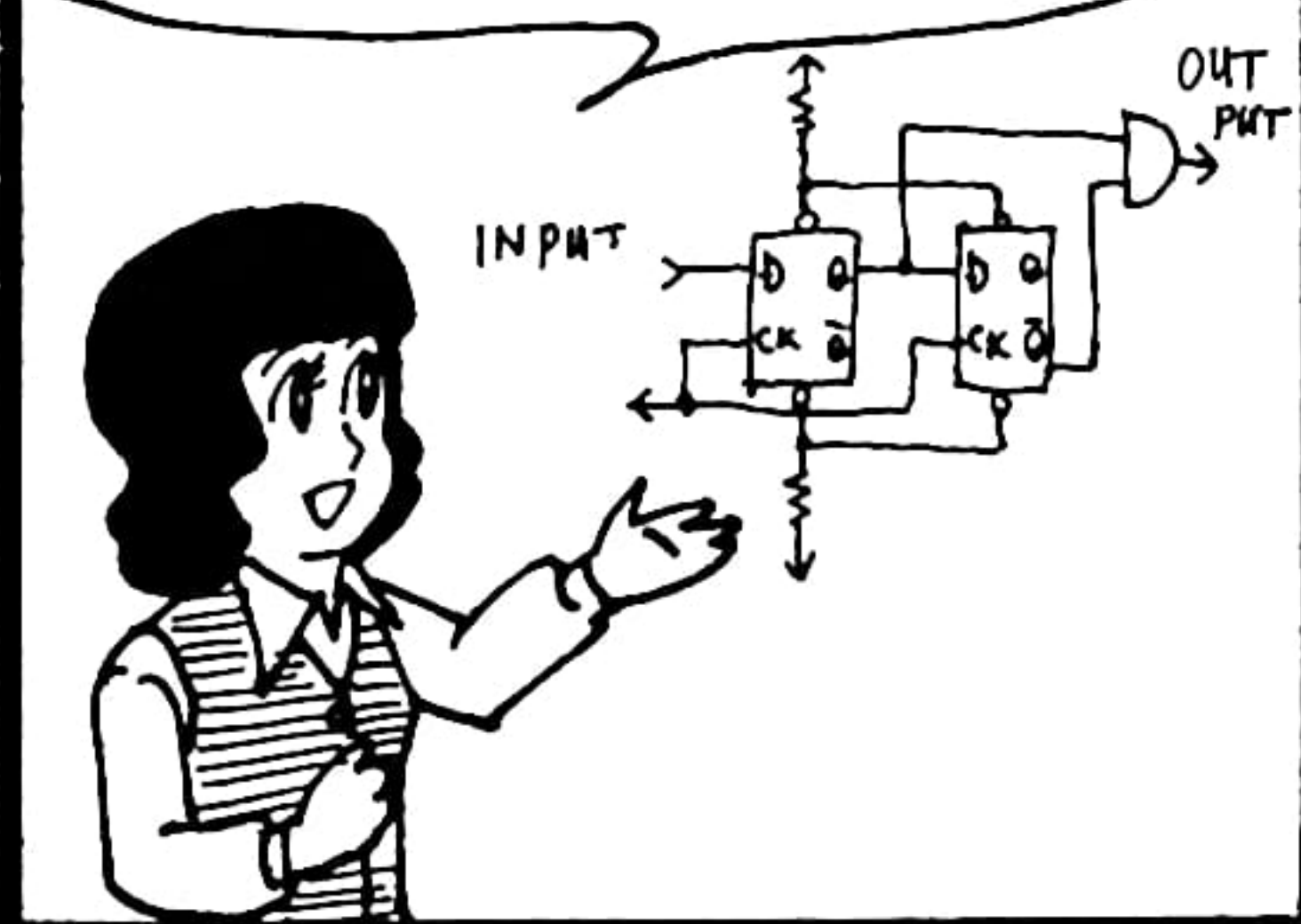
入力 ㋑

クロック ㋺

VCC

このタイミングチャートを見てもわかるようにクロックのエッジにかからない入力は無視されることになり、ノイズに対しても強い回路が設計できます。

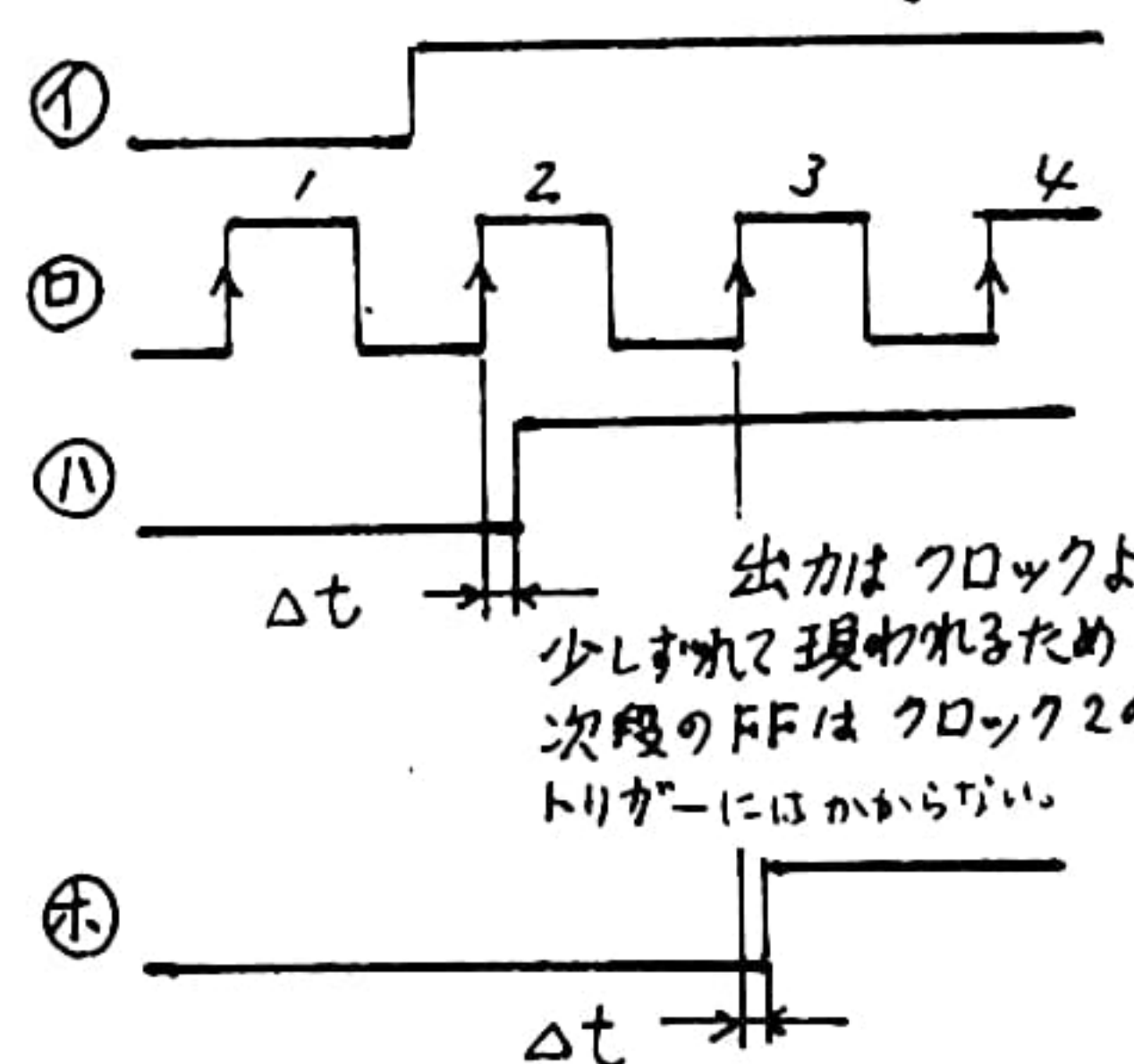
D CK Q Q̄

入力１は クロックのエッジにかからず出力に現われていない。

エッジトリガーＦＦ応用の基本です。よく、読み返して理解してください。覚えておけば回路図を読むのが楽になりますよ。

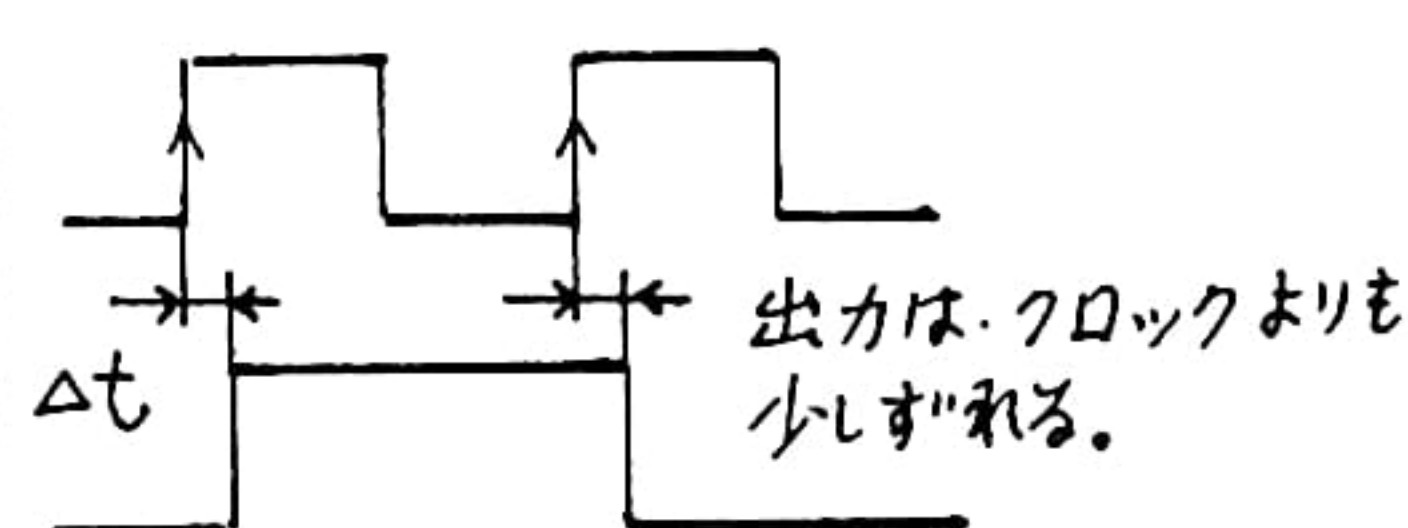
INPUT OUTPUT

これは、エッジトリガーだから、直列につなぐと１クロック分ずれることを応用したものです。

㋑ ㋺ ㋩ ㋭

Δt

出力は クロックより少しずれて現われるため次段のFFは クロック２のトリガーには かからない。

前のコマは、入力パルスの最初の方で１クロック分のパルスが得られることを示しています。

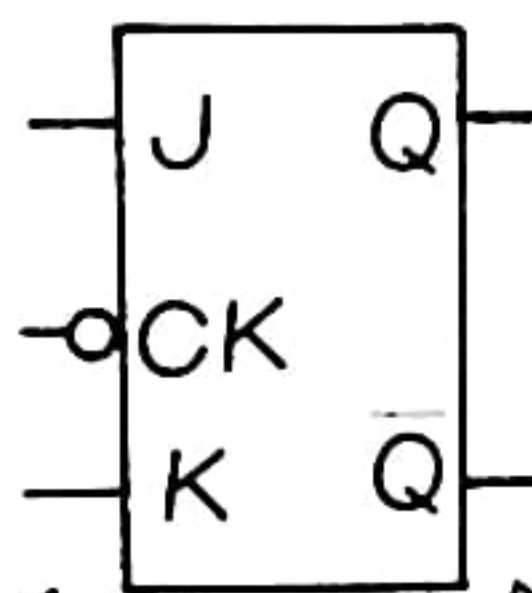
Δt

出力は クロックよりも少しずれる。

クロック	インプット		アウトプット	
CK	J	K	Q	$\overline{Q}$
↓	L	L	QO	$\overline{QO}$
↓	H	L	H	L
↓	L	H	L	H
↓	H	H	TOGGLE	

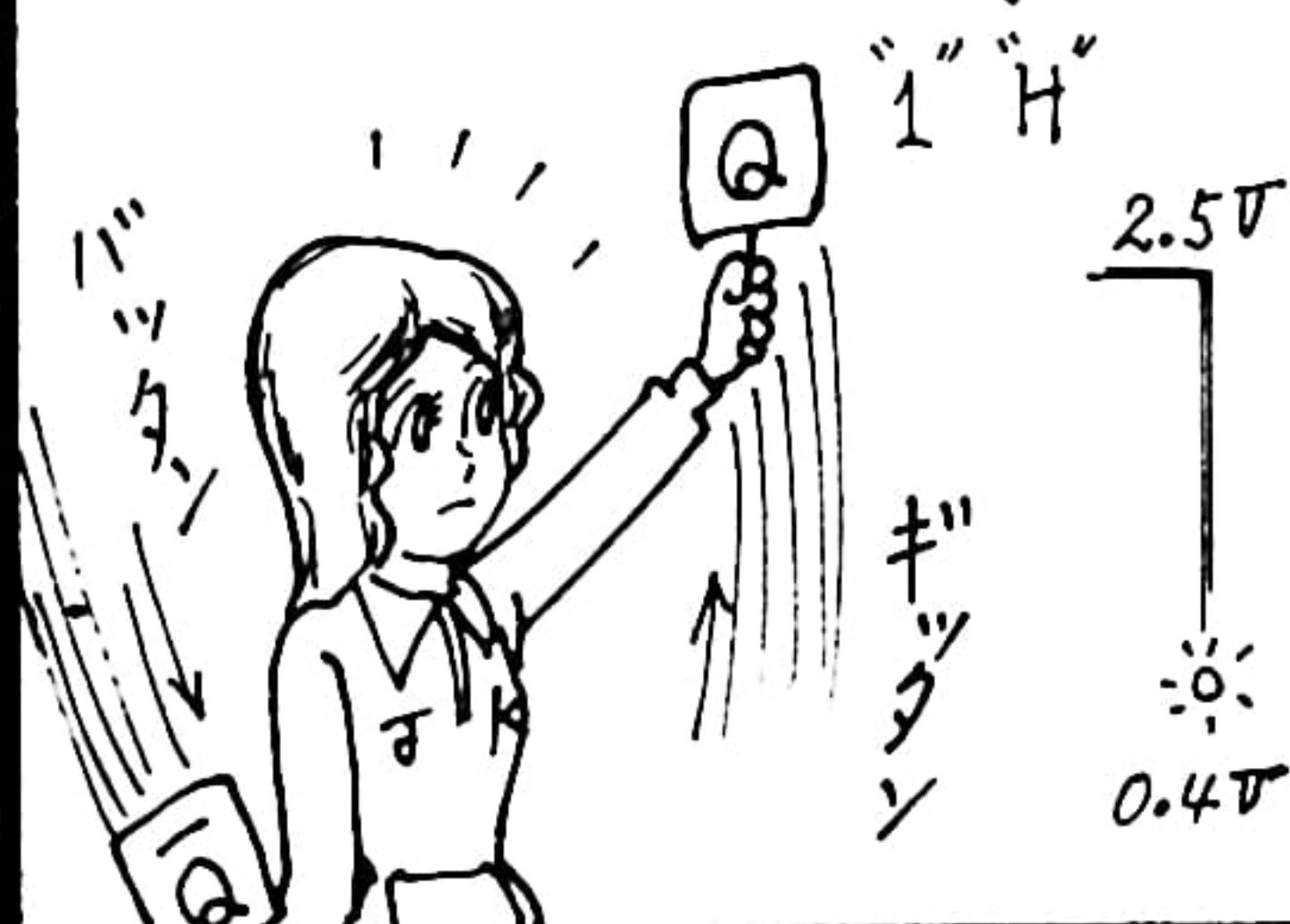
クロックが↓となる前の状態を保持します。

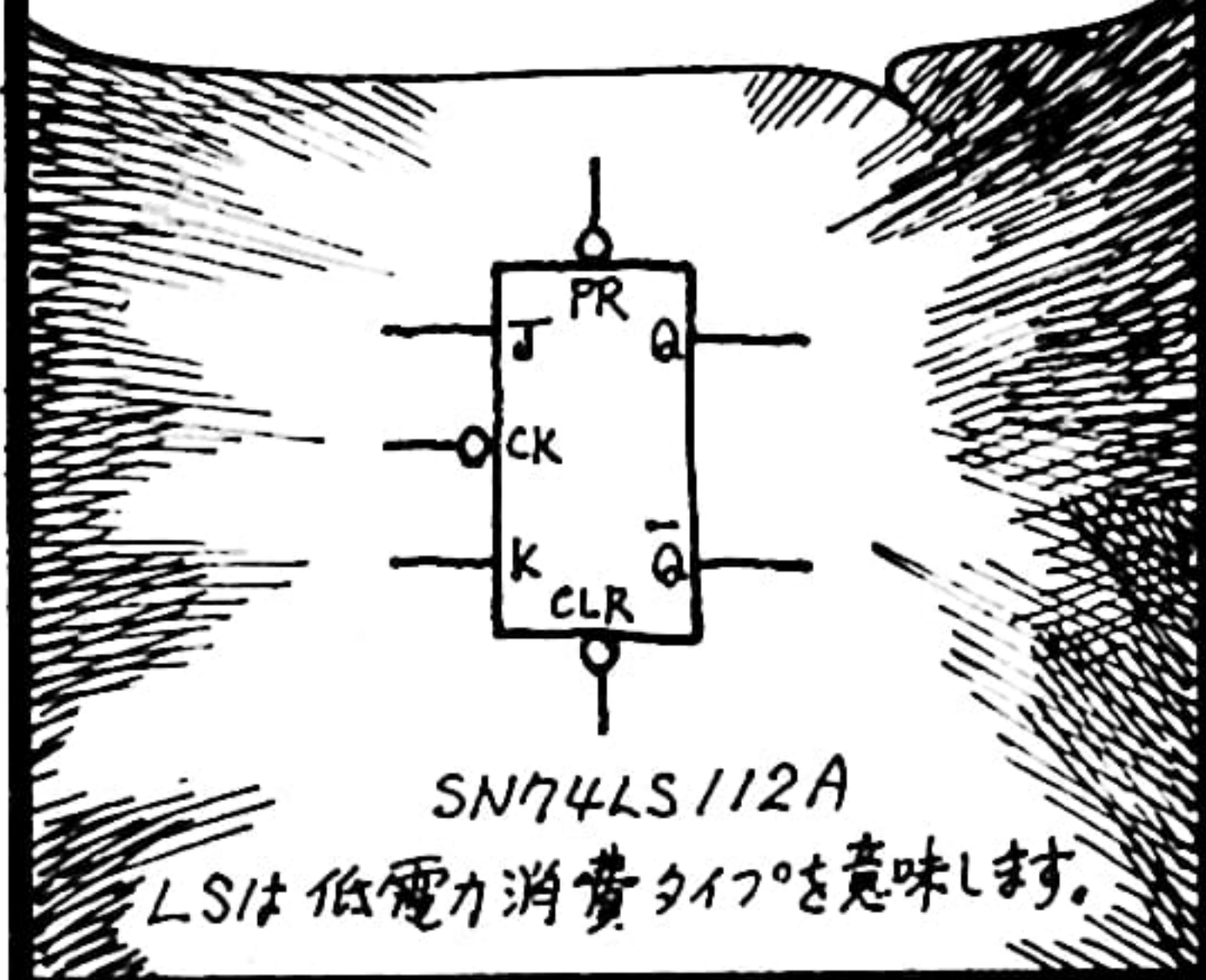
JK型はセット入力とリセット入力を持ったFFです。

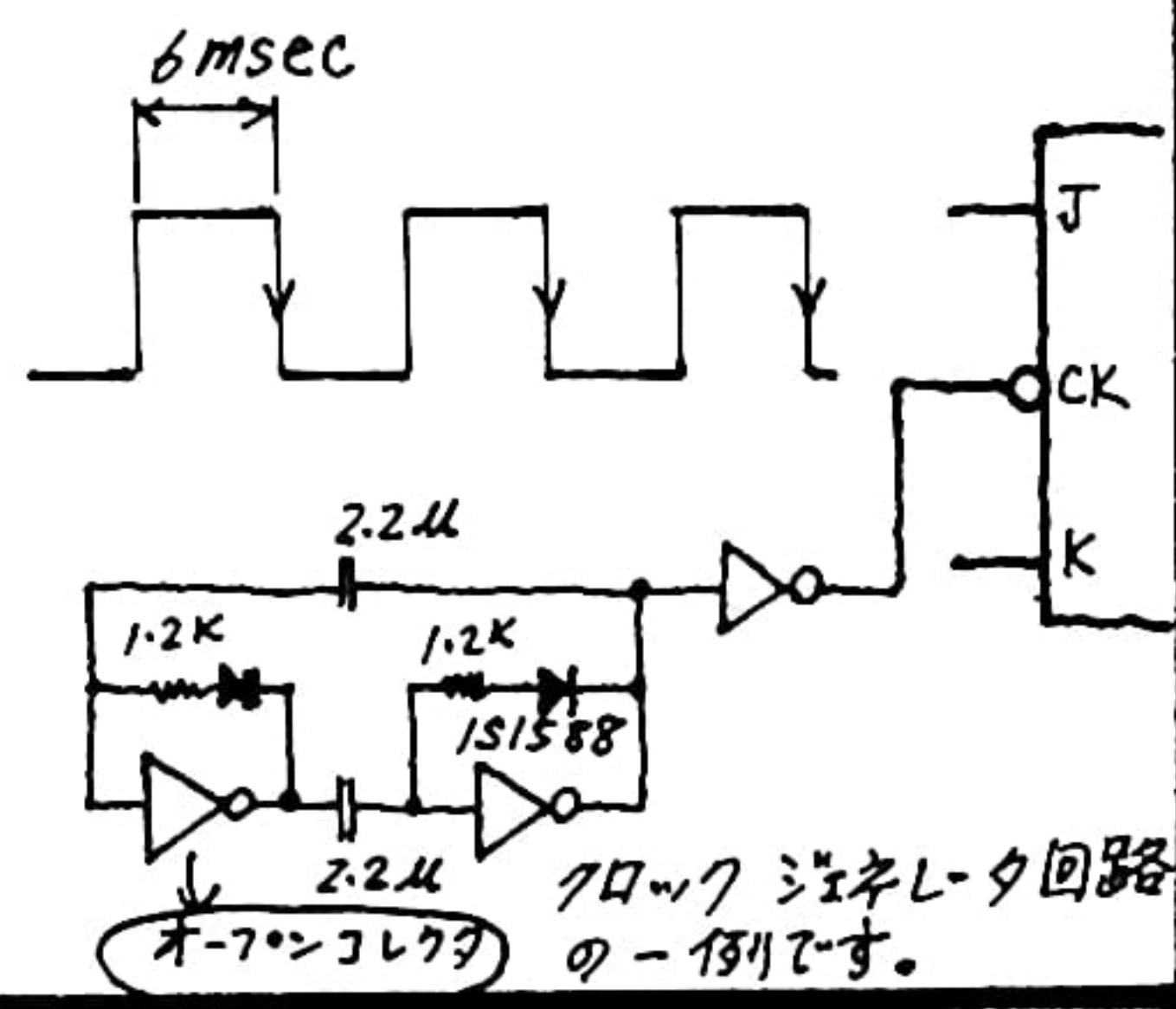
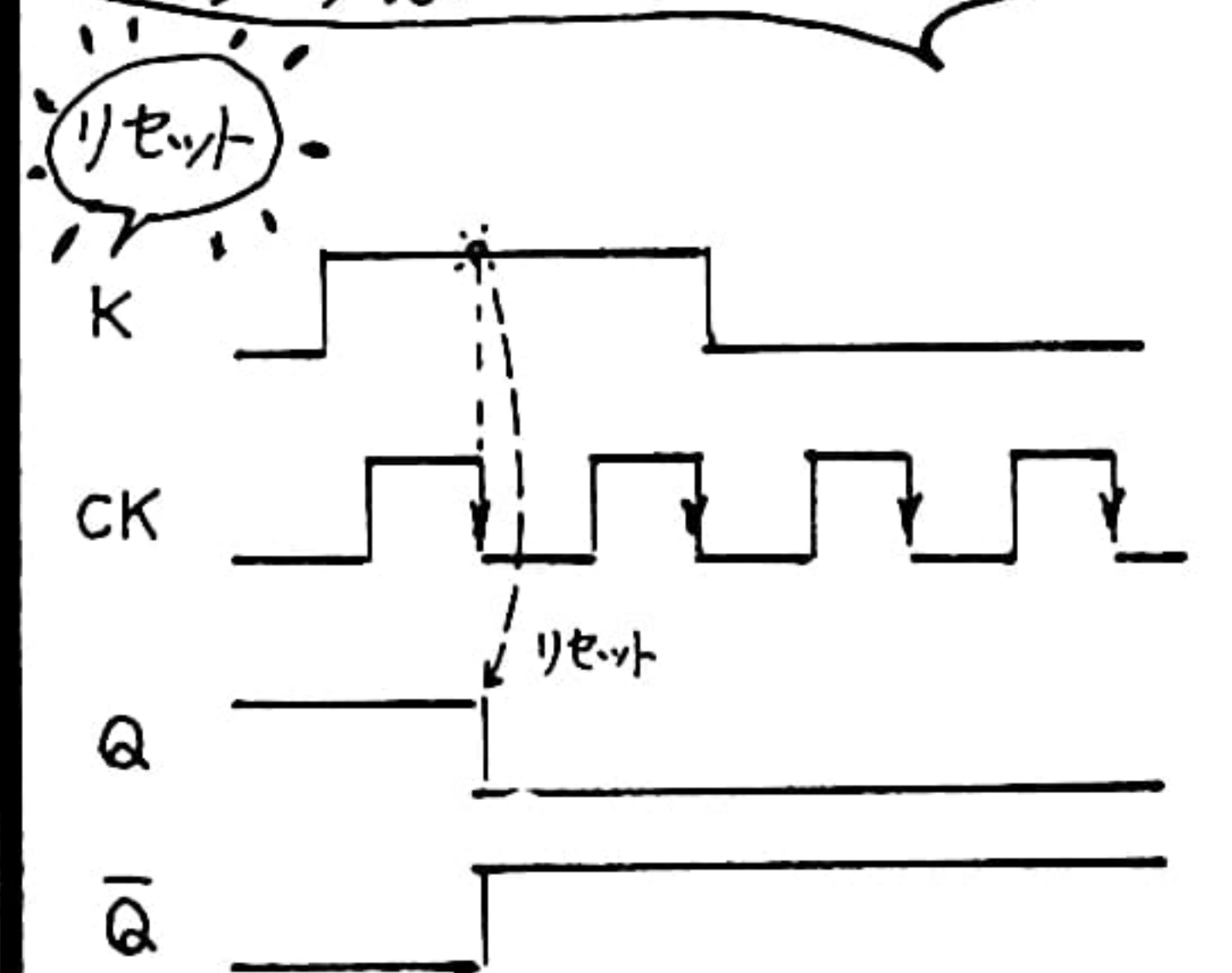
JK型
FF

J Q
CK
K $\overline{Q}$

クロックが↓となる前の状態から反対の状態に変わります。

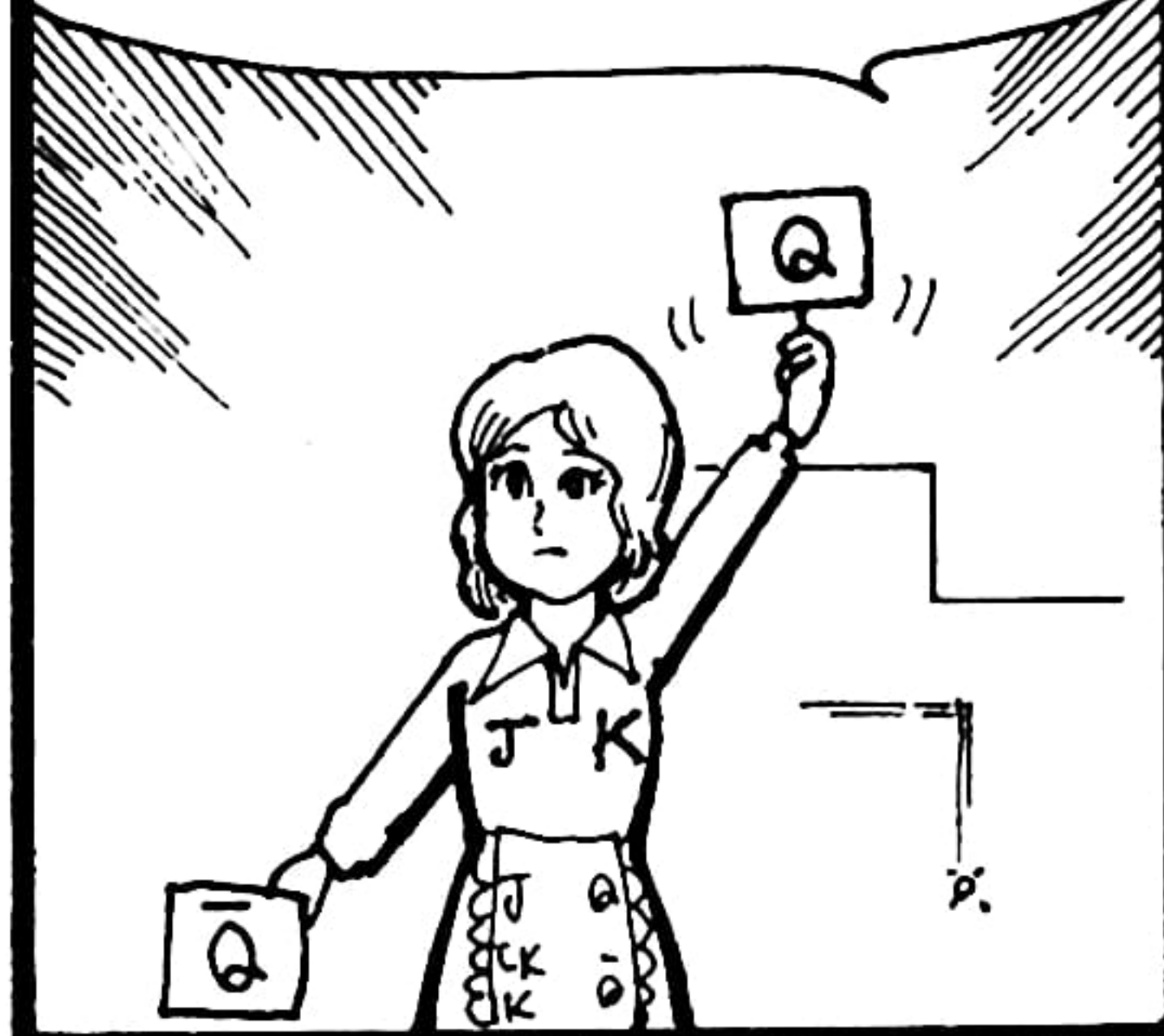
"セットする"とは出力QをHにすることだと考えてください。$\overline{Q}$はQの反対ですからLになります。

バッタン
Q
"1" "H"
2.5V
ギッタン
0.4V

JKにもいろんな種類（トリガーが正エッジのものもある。)のものがありますが、ここではSN74LS112Aを例にとってみます。

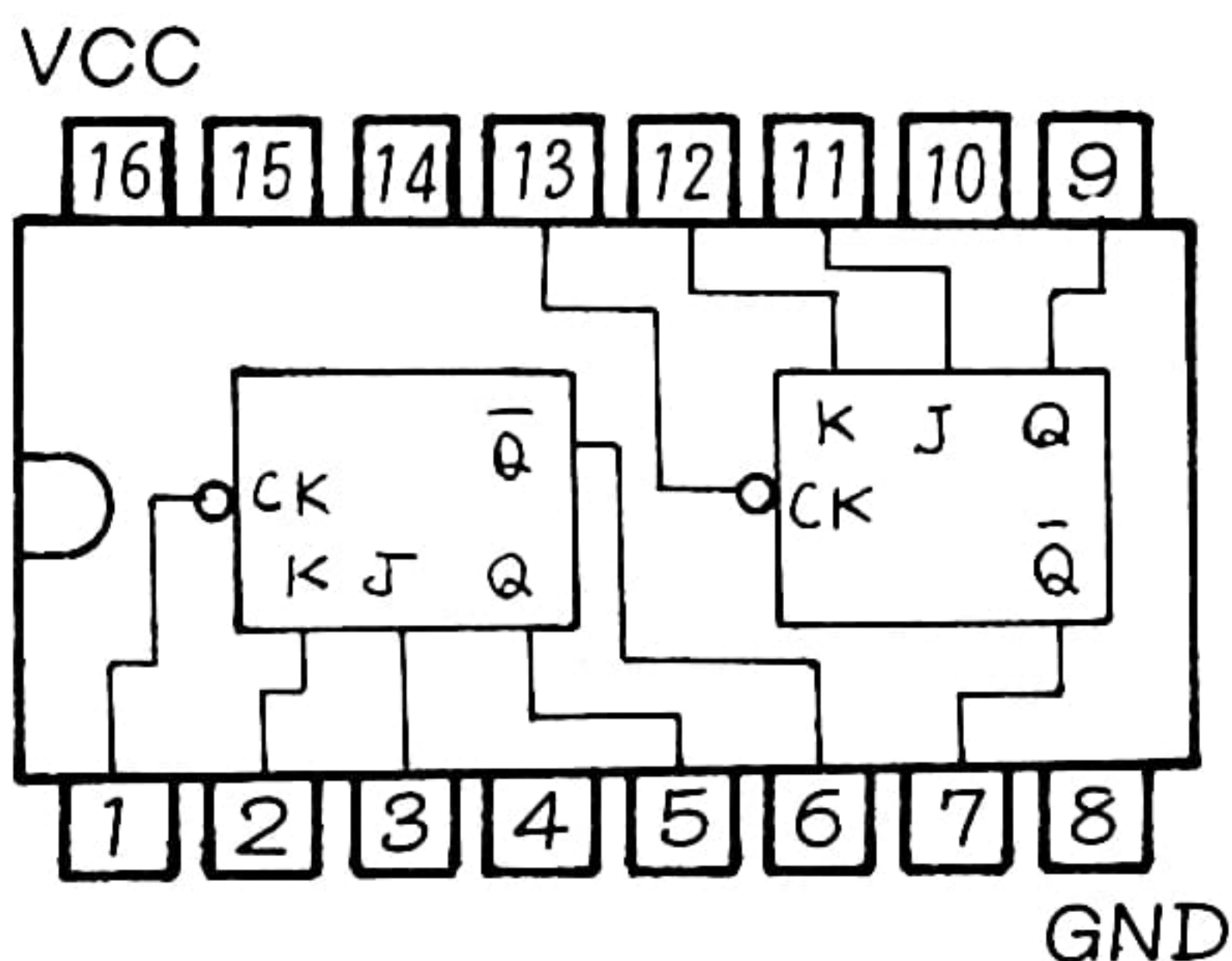
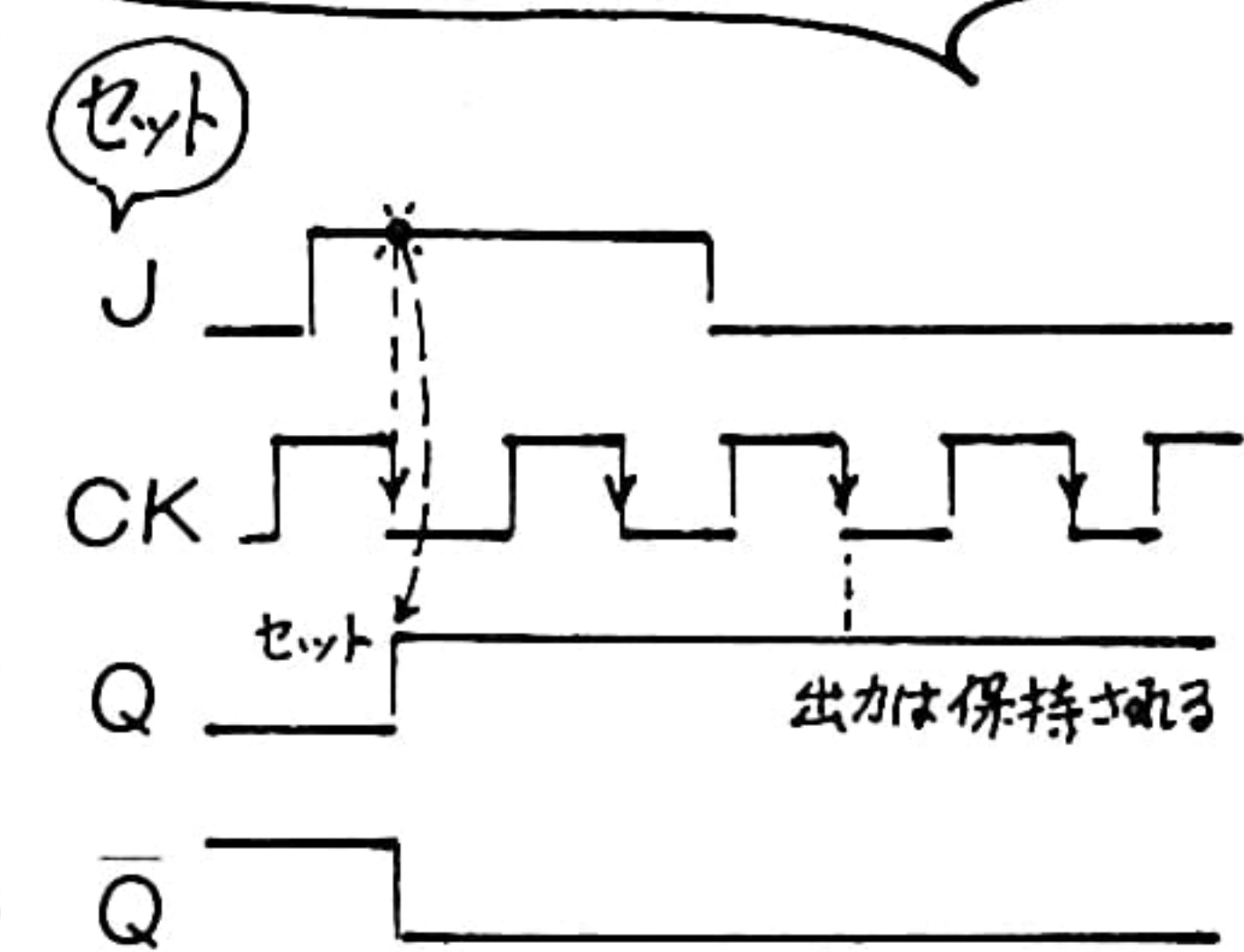
PR
J Q
CK
K $\overline{Q}$
CLR
SN74LS112A
LSは低電力消費タイプを意味します。

出力を変化させるタイミングをとるクロックのエッジは負側です。

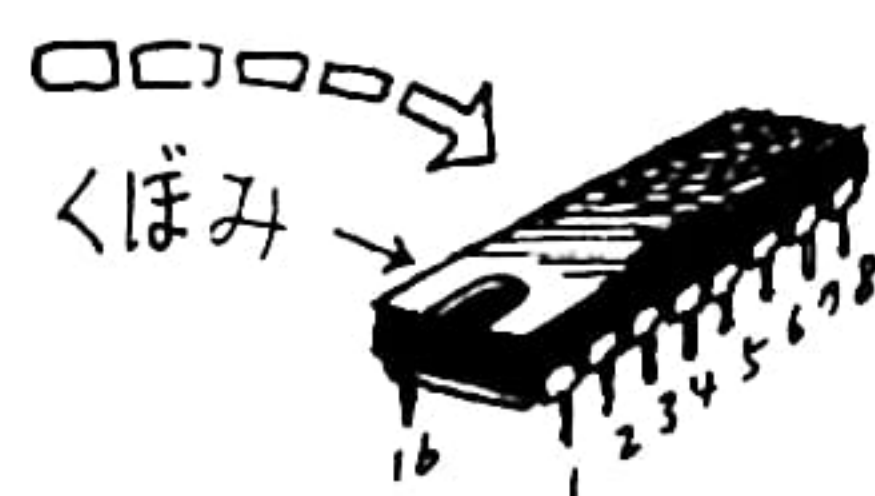
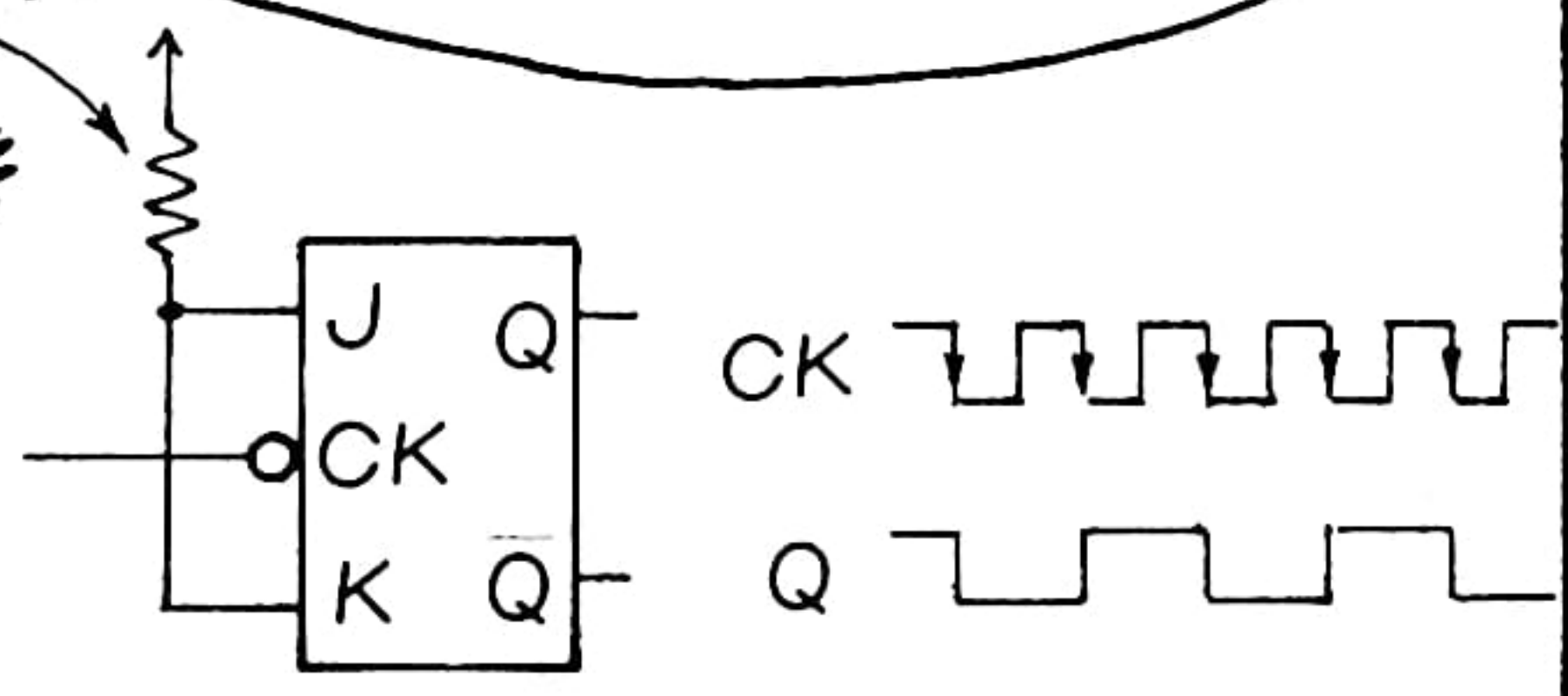
6msec
J
CK
K
2.2μ
1.2K
1.2K
1S1588
2.2μ
オープンコレクタ
クロックジェネレータ回路の一例です。

そして、出力Qがリセット入力Kによって、Lに落ちるタイミングは……

リセット
K
CK
リセット
Q
$\overline{Q}$

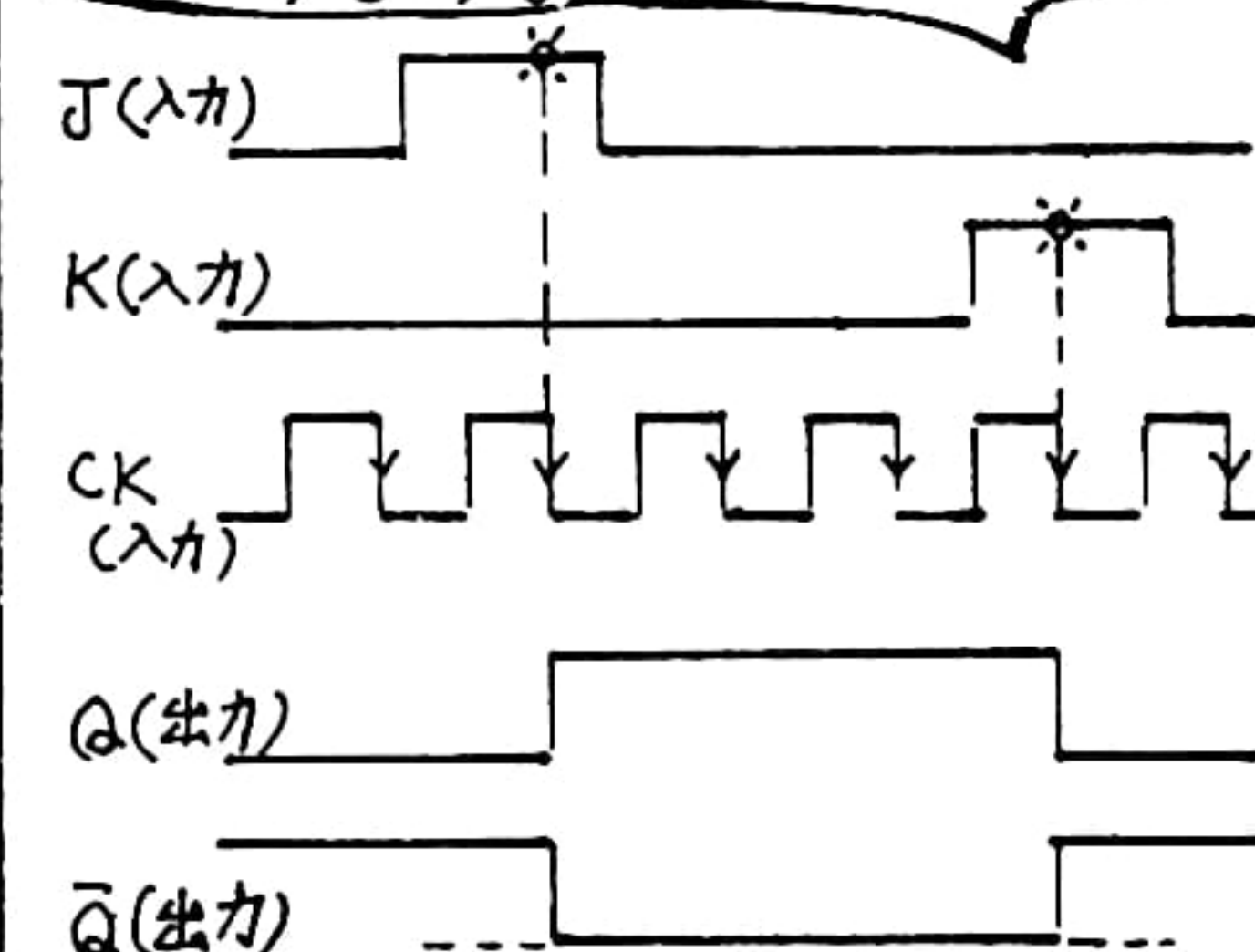
入力パルスがLに落ちてからも出力QはHに保持され続けます。

Q
JK
$\overline{Q}$

まずQが入力Jによってセットされるタイミングを見てみましょう。

セット
J
CK
セット
Q
出力は保持される
$\overline{Q}$

VCC
16 15 14 13 12 11 10 9
CK $\overline{Q}$
K J Q
K J Q
CK $\overline{Q}$
1 2 3 4 5 6 7 8
GND

JKもふつう2個がひとつのパッケージに入っているものが売られています。SN74LS112Aは16ピンです。SN7474は14ピンでした。

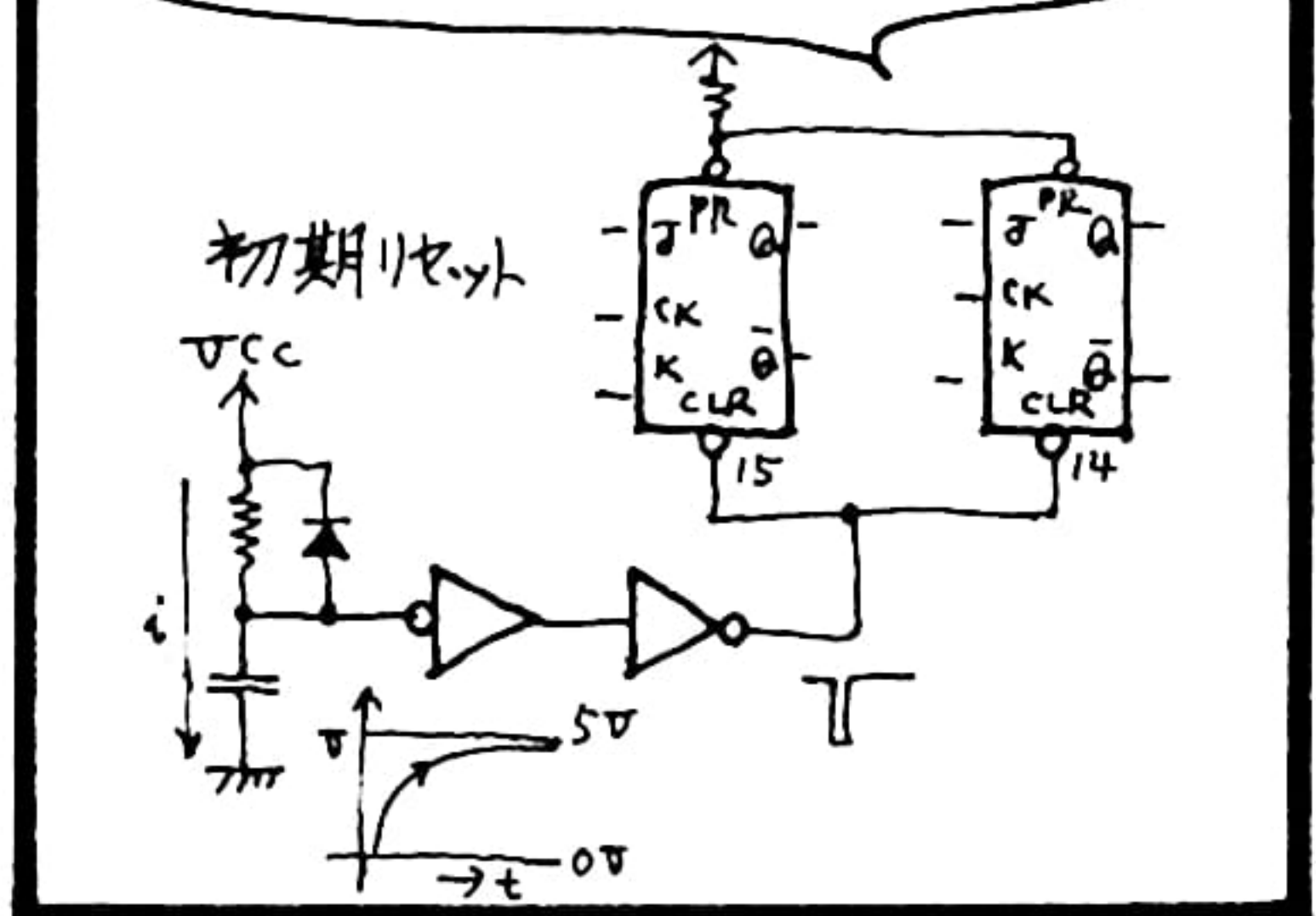
くぼみ

このセット・リセットをひとつにまとめるとこのようになります。

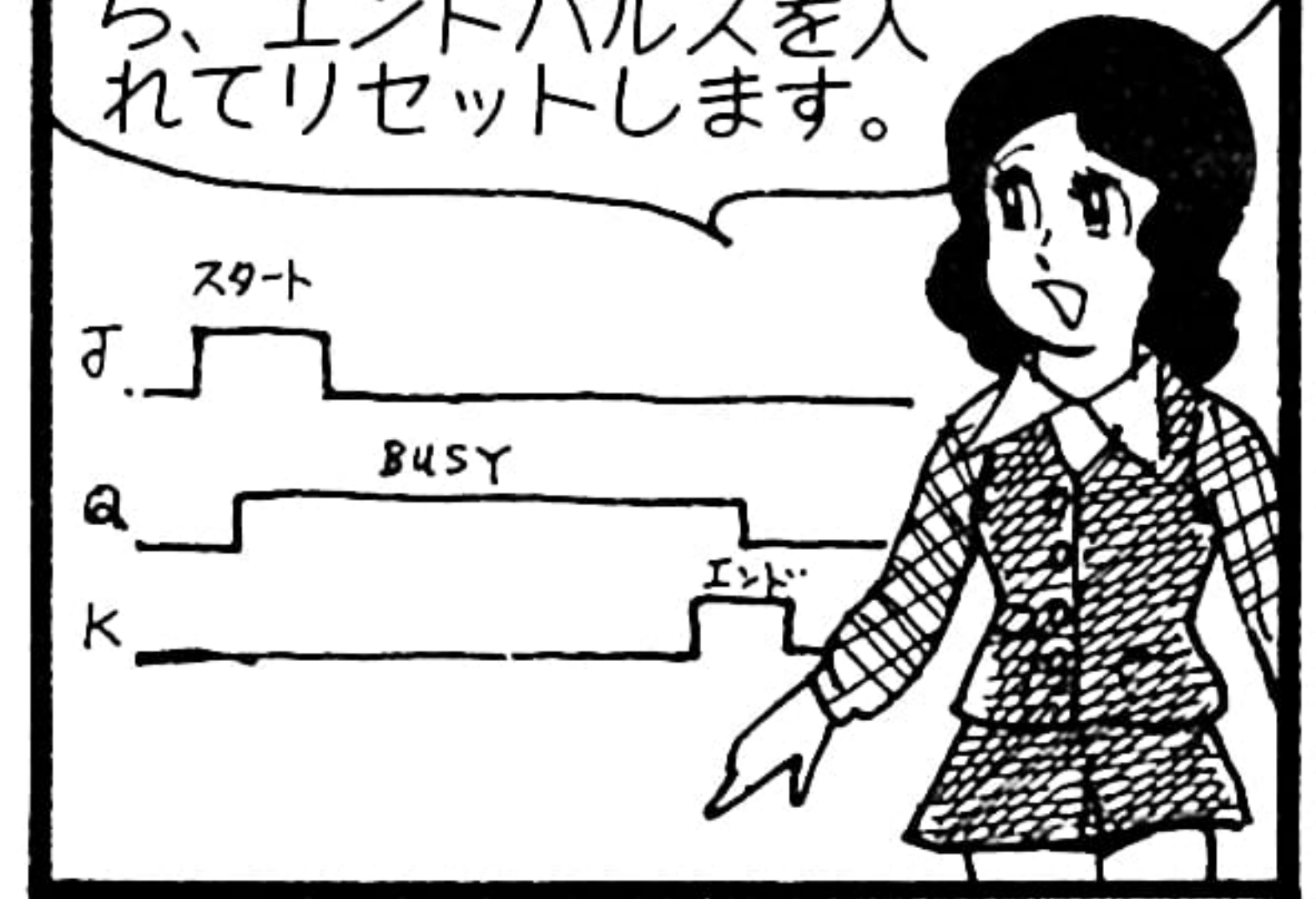
J(入力)
K(入力)
CK(入力)
Q(出力)
$\overline{Q}$(出力)

ＪＫ入力をこのようにプルアップすれば両方共いつもＨなので出力Ｑにクロックの２倍周期の波形が得られます。

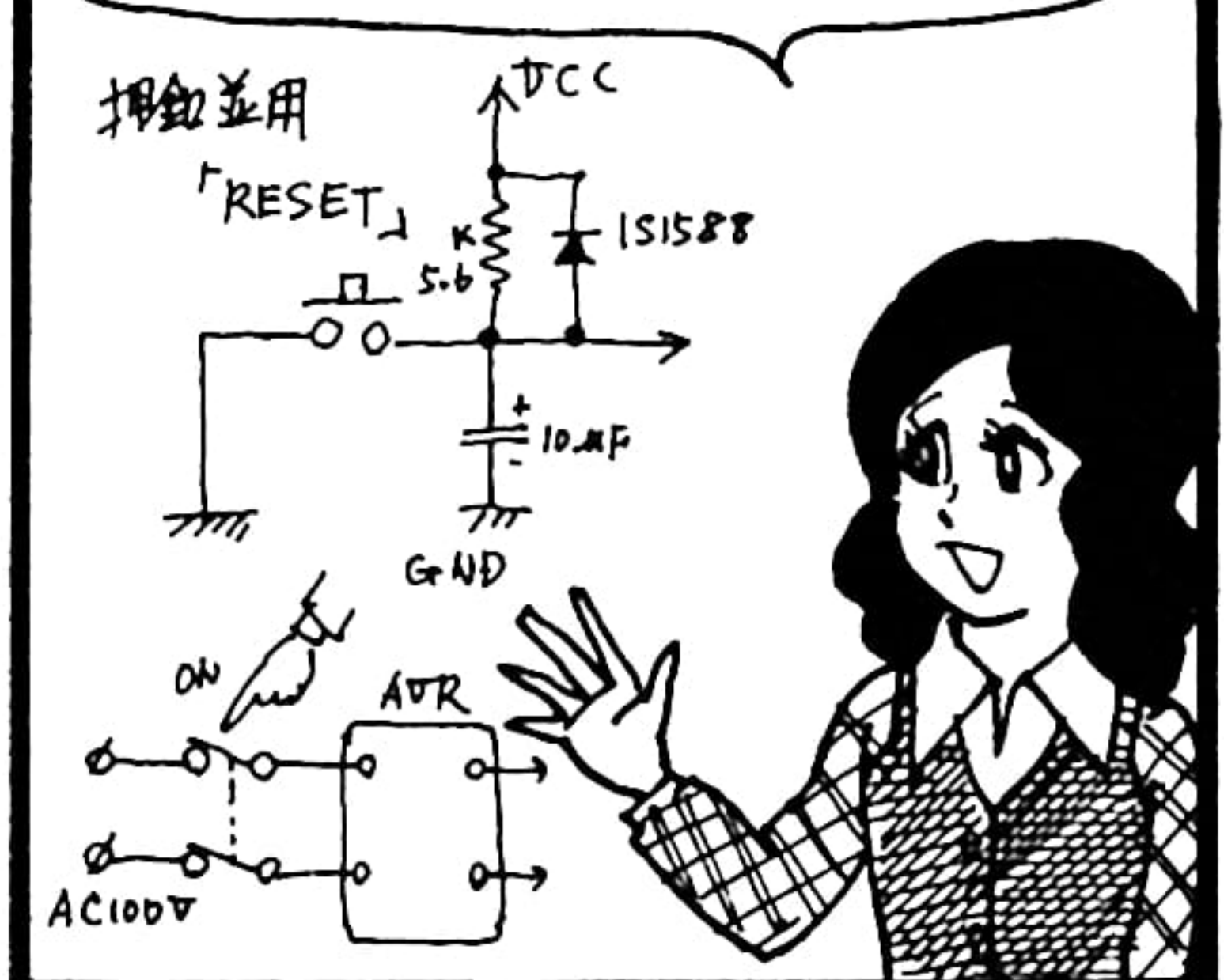
プルアップ抵抗

J Q CK K Q̄

CK

Q

このJK型ＦＦをどう使うかはあなたの自由です。

好きにしてちょうだい

ＪＫＦＦの場合は電源投入時、Ｑ．Q̄のどちらかがＨにセットされるかわからないので電源リセット回路をつけておく必要があります。

初期リセット

たとえば、印字ボタンを押すことは、スタートパルスのきっかけをつくることですし、この場合のＢＵＳＹは印字中に当たります。

印字中

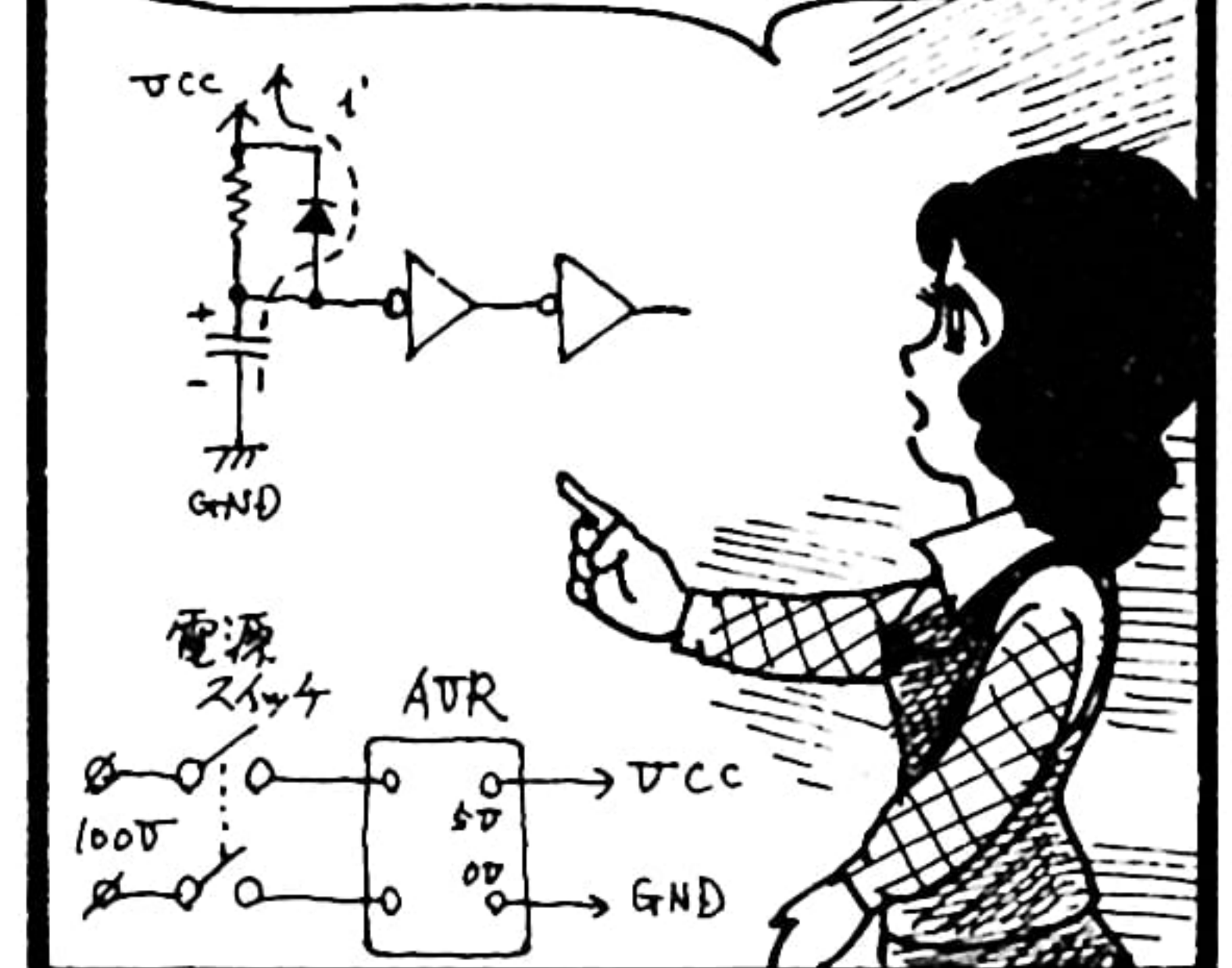
印字

プリンタ

ＪＫＦＦのふつうの使い方の例としてはＢＵＳＹ（動作中）信号などがあります。Ｄ ＦＦでつくったスタートパルスを使いＢＵＳＹ状態にし、作業が終わったら、エンドパルスを入れてリセットします。

スタート

BUSY

エンド

これは電源リセット回路の定石ですから覚えておいてください。

押釦並用 「RESET」

1S1588

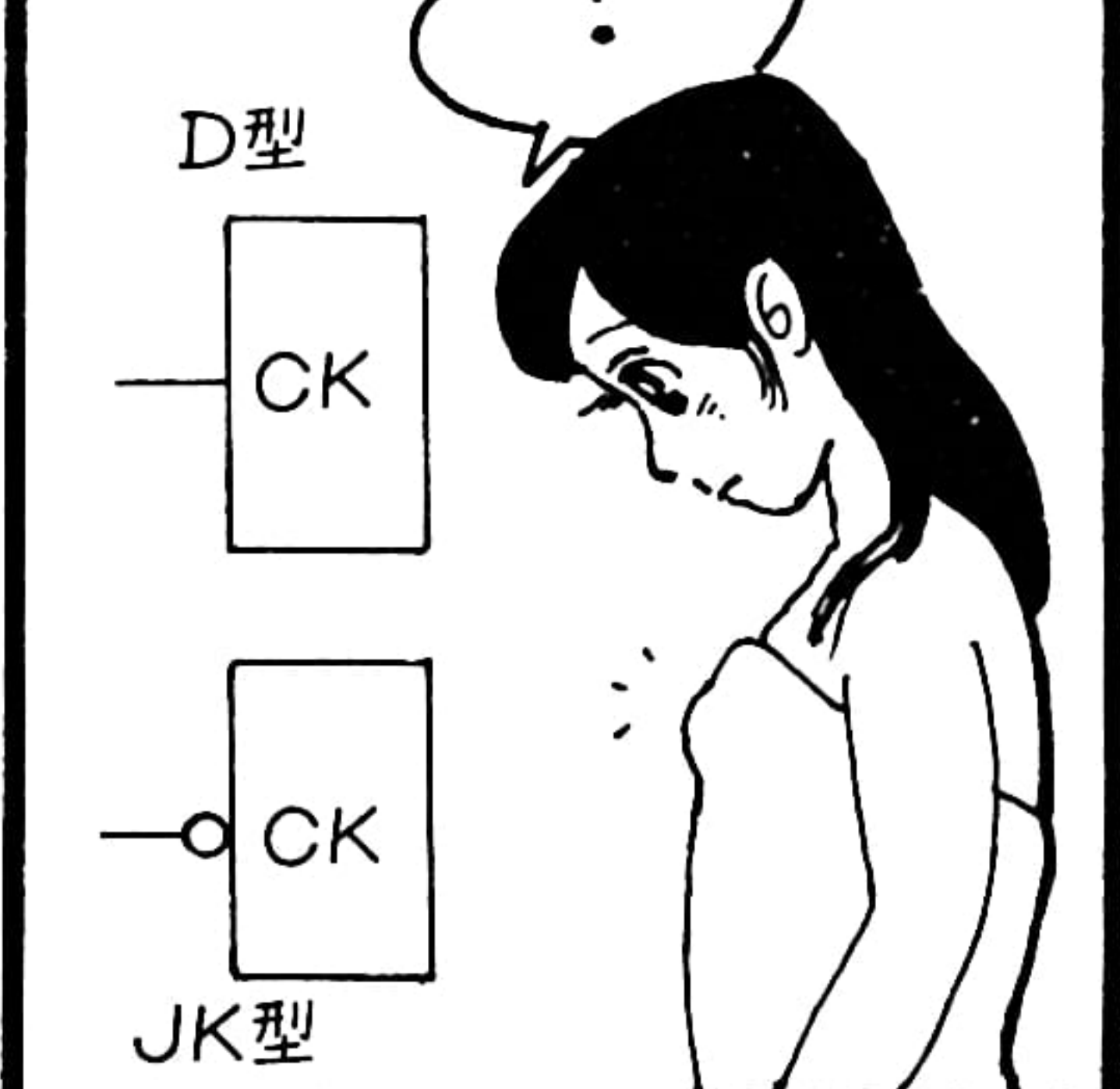
10μF

GND

AVR

AC100V

これをつけていないと、次に電源を投入した時リセットが効かなくて回路が変な働きをすることになります。

？

このダイオードは、電源をOFFした時コンデンサの電荷を逃がすためのものです。

GND

電源スイッチ

AVR

100V

GND

エッジトリガー　ＦＦにどうして正負のものがあるのでしょう？わざわざ２種のトリガーがあるのには、何か意味があるはずです。

D-FF　JK-FF

？

D型

CK

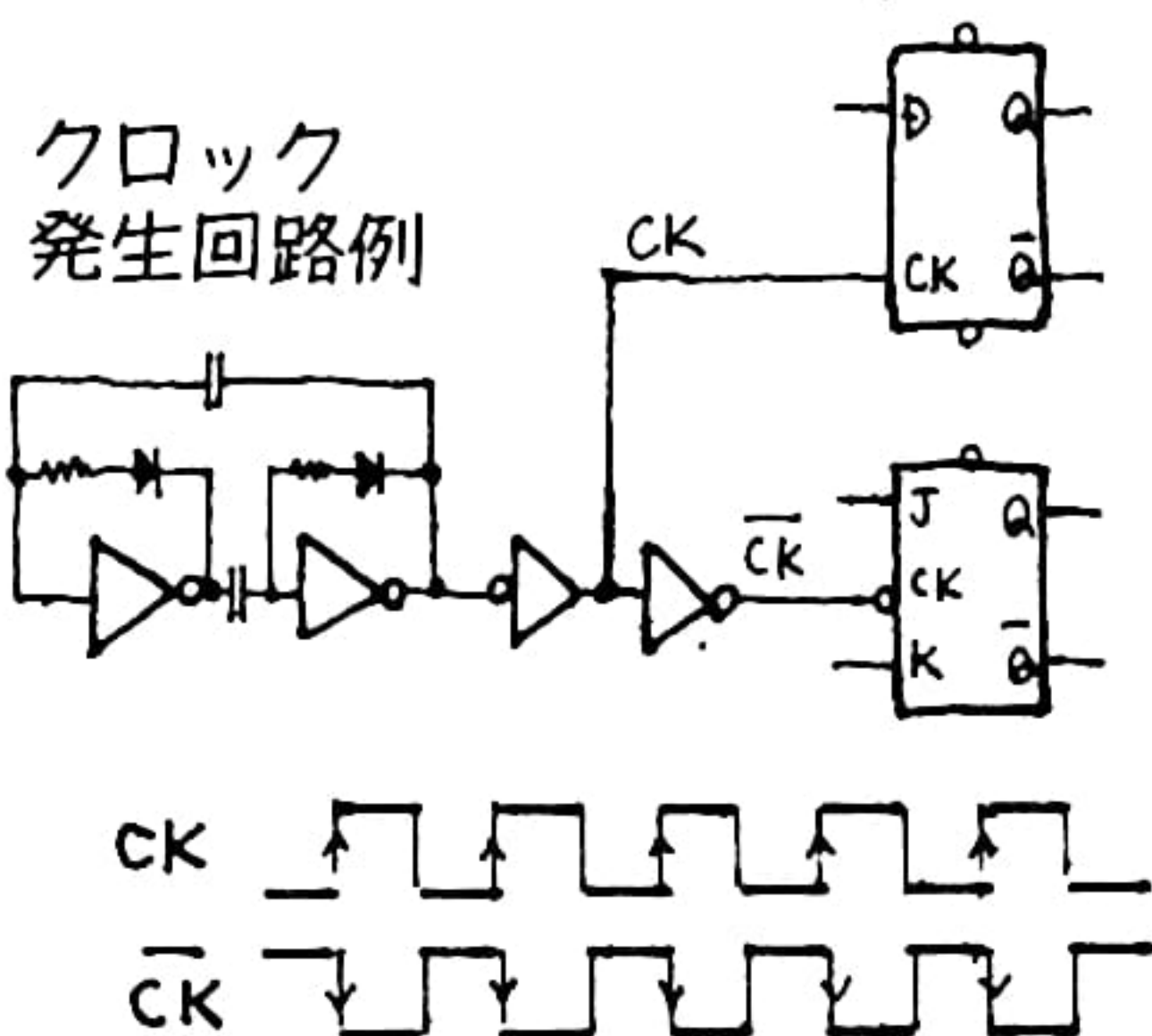
CK

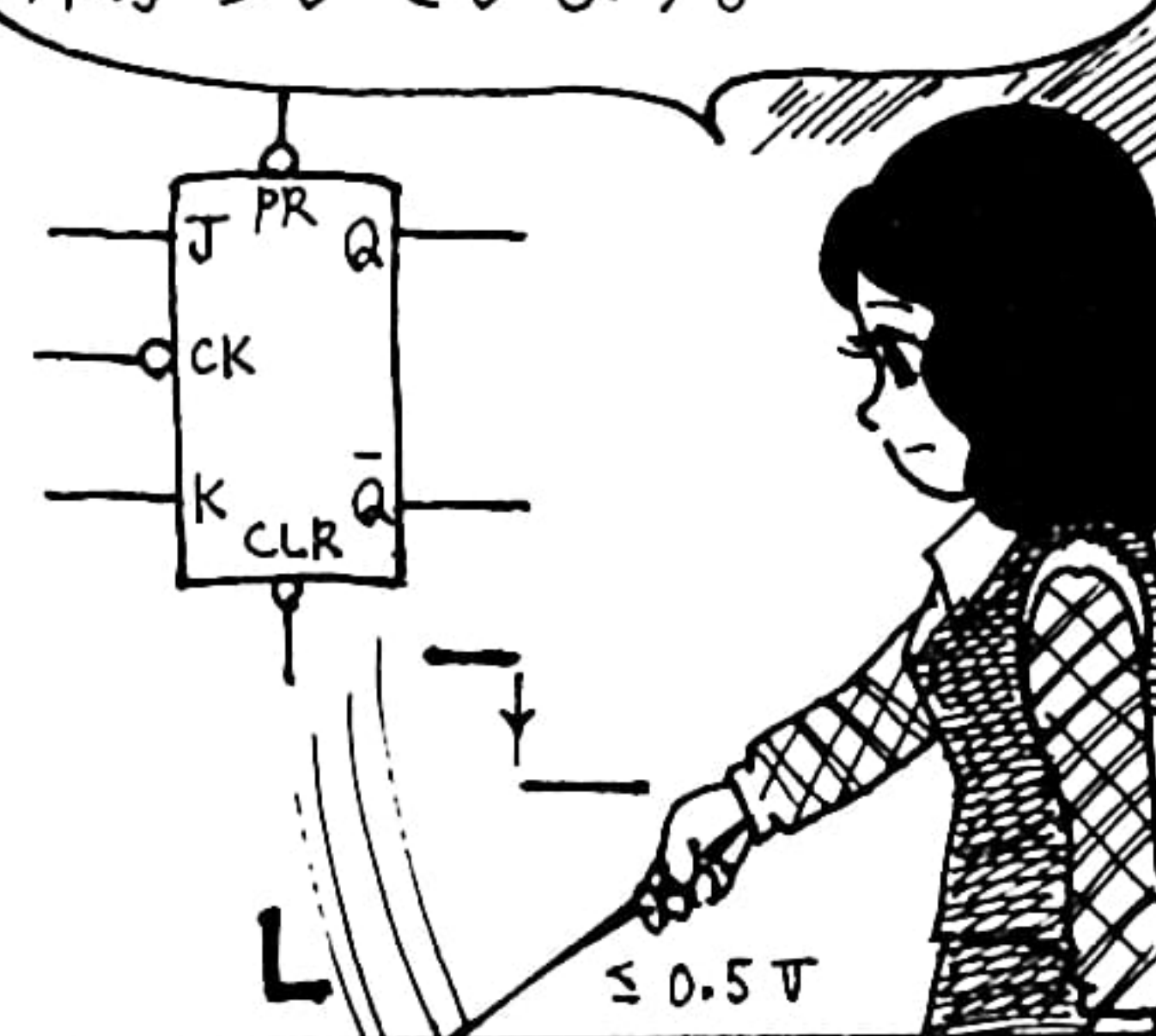
JK型

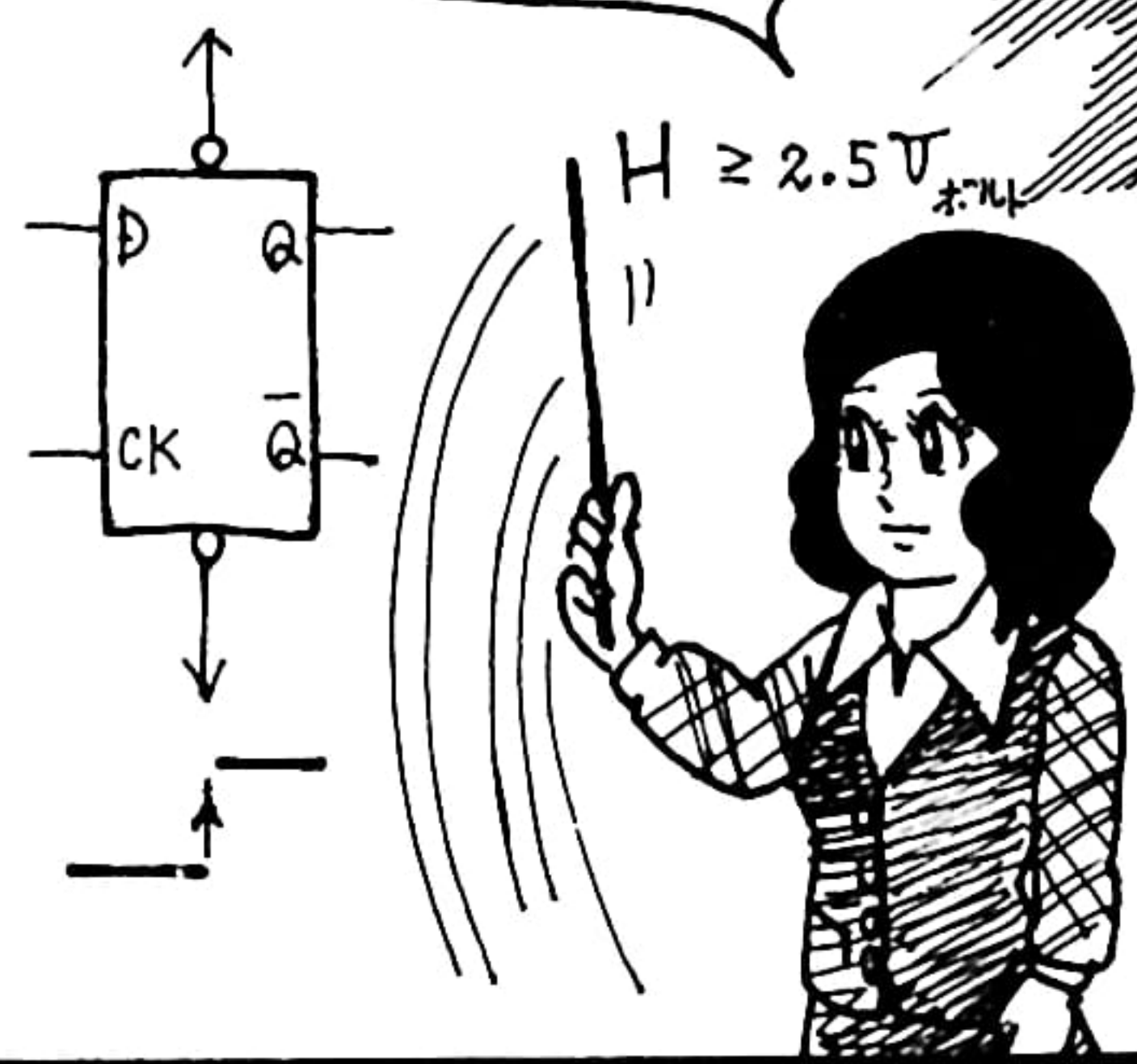
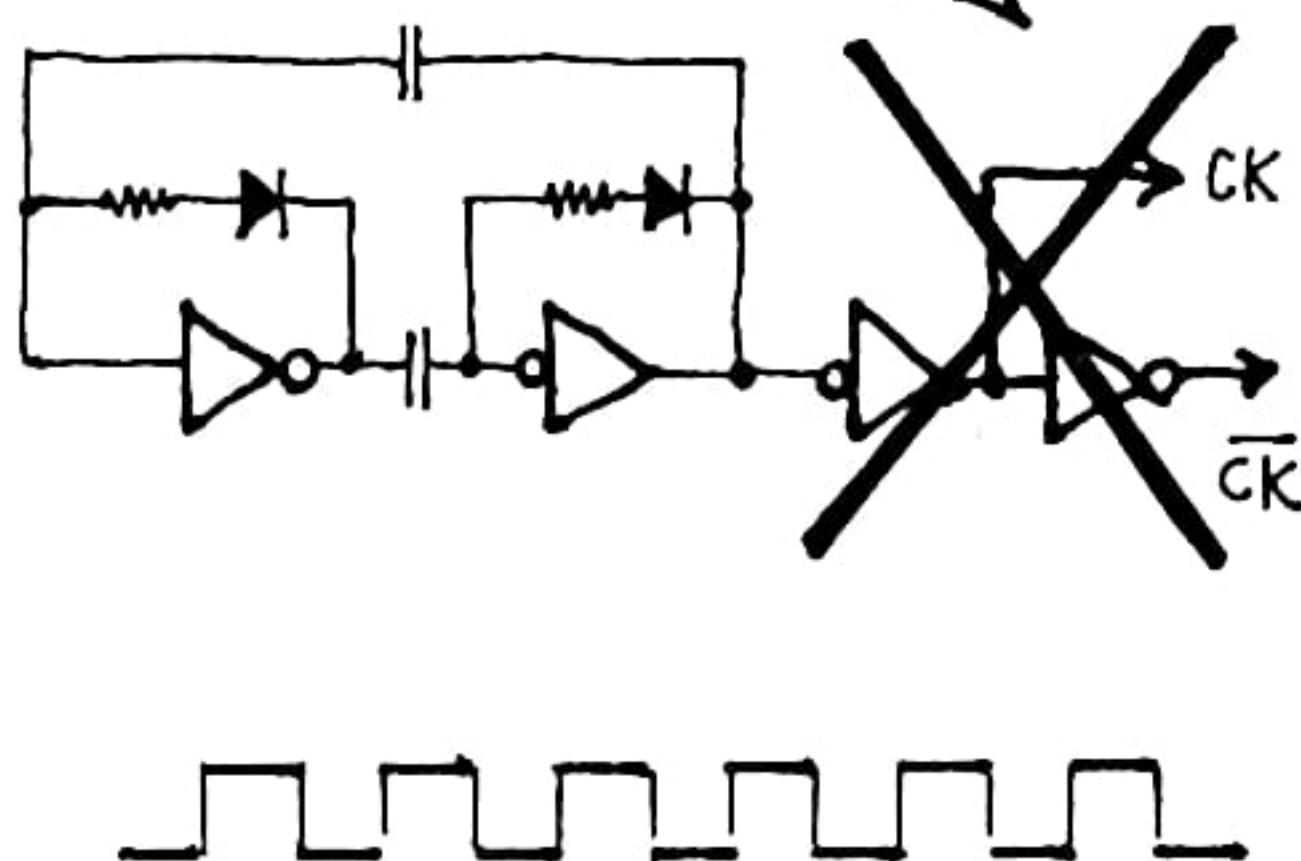
DとJKFFをうまく組み合わせると、ノイズに強いダイナミックな回路を設計できます。

これは単純に考えると、"DとJK FFのクロックはCKと$\overline{CK}$の関係がある"と思ってしまうかもしれません。

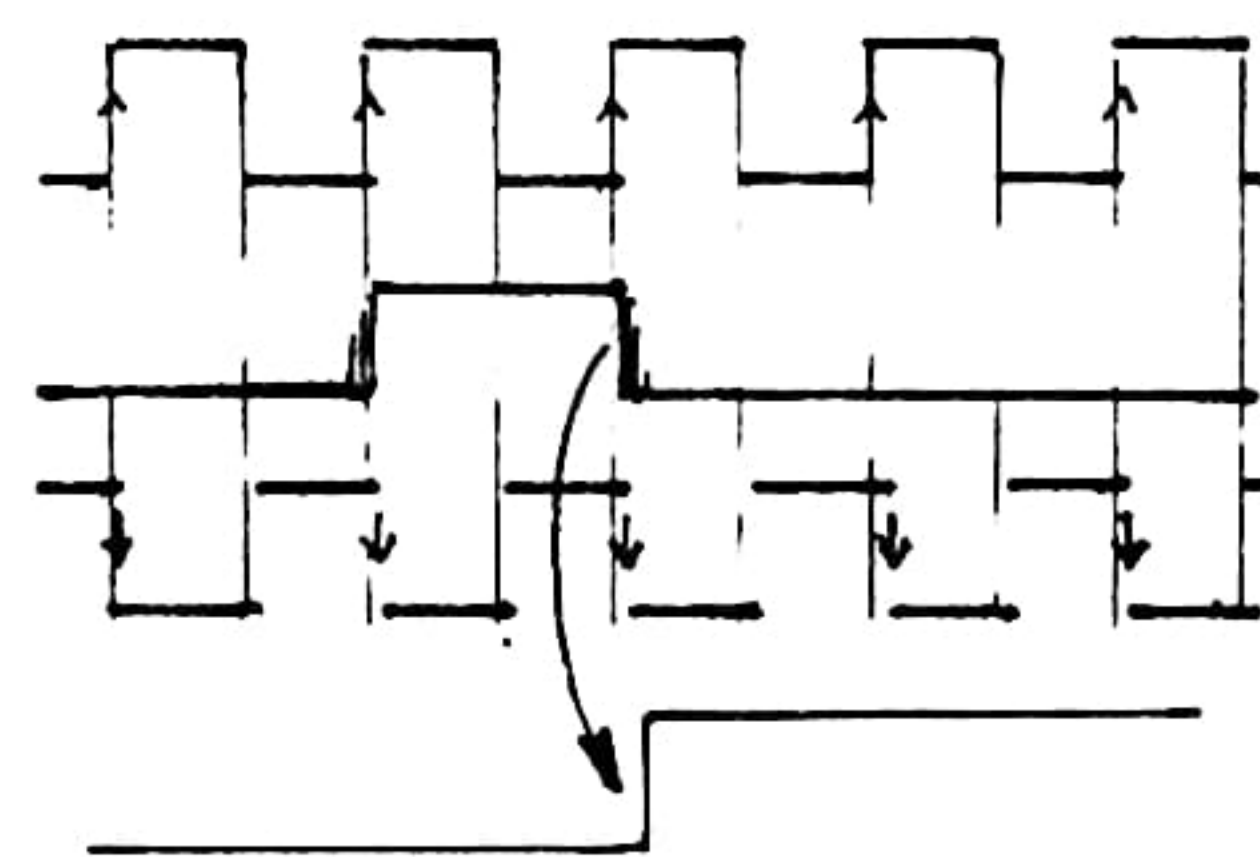
クロック発生回路例

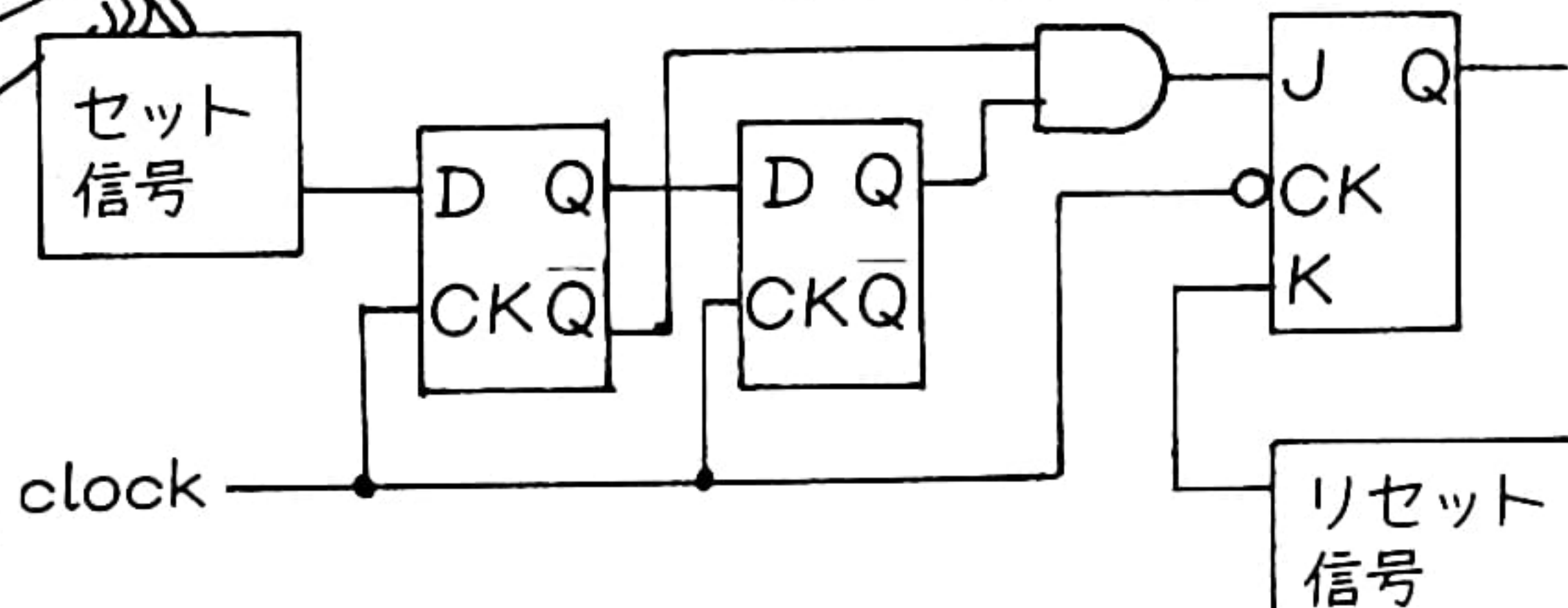
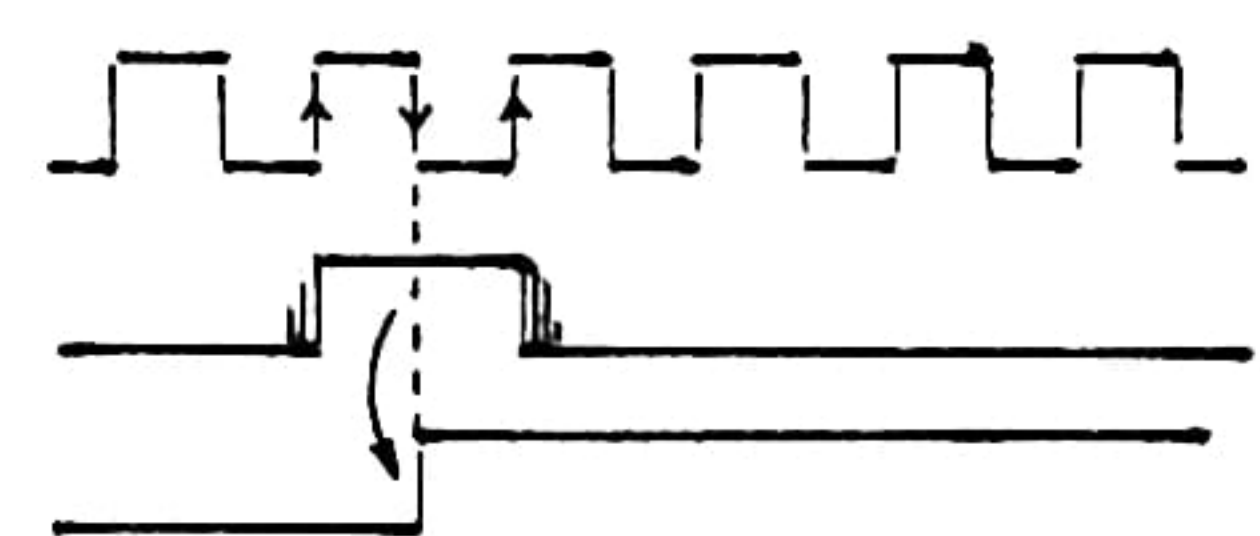
CK
$\overline{CK}$

JK FFはクロックの立下りでトリガーがかかりました。こっちのCKには例の小さな○印がついています。

L
≦0.5V

D FFはクロックの立上りでトリガーがかかり、

H ≧2.5V ボルト

クロックをCK・と$\overline{CK}$にするのは考えたようで考えてないのです。単なる早トチリです。

配線距離が長かったりすると、遅延や反射などでパルスの両端は、きわめて不安定な状態ですからこのような回路は信頼性がありません（回路がまったく働かないということはありません）。

これは失敗談です。

CKでつくったパルスを$\overline{CK}$を入れたJK FFに入れると図のようにきわどいタイミングです。

フリップフロップは1ビットのメモリと考えられます。メモリは安定域で読み書きすることが原則です。

セット信号
clock
リセット信号

真上のコマとこのコマのタイミングの違いがわかりますか。パルスのまん中にクロックの↓がきています。

チップの中の回路はこのようになっています。入力Bが↑になるとCとRで決まる長さで1パルス出力されます。

SN74121

インプット			アウトプット
A1	A2	B	Q
L	L	↑	⎍

A1 A2 B

④ワンショットパルサー

T
$T = C_T \cdot R_T \log 2e$

押釦はパネルの表面に取り付けられていますからプリント基板まではちょっと距離がありますから、ノイズが問題になってきます。

プリント基板

これは－121を使ったスタートパルス発生回路の一例です。押釦のチャッタリングによる誤動作防止のため－121の特徴を利用しています。

Vcc ① 5 10 11 9 B －121 3 A1 4 A2 6 ② D Q －74 CK Q̄ D Q －74 CK Q̄ ③ ④ クロック

スタートパルス発生回路のタイミングを見てみましょう。

① T T ② チャッタリングによって離した時にもワンショットがかかる クロック ③ ④

スイッチを押した時やリレーの接点が入った時、またそれらの逆の時、短い時間ですが信号はON、OFFをくり返します。これをチャッタリングというのです。

Vcc ① 60msec 60msec チャッタリング

抵抗が2KΩから40KΩ、コンデンサが10pFから10μFまでなら、$T=C_T \cdot R_T \log 2$
$\fallingdotseq 0.7C_T \cdot R_T$
という関係が成立します。

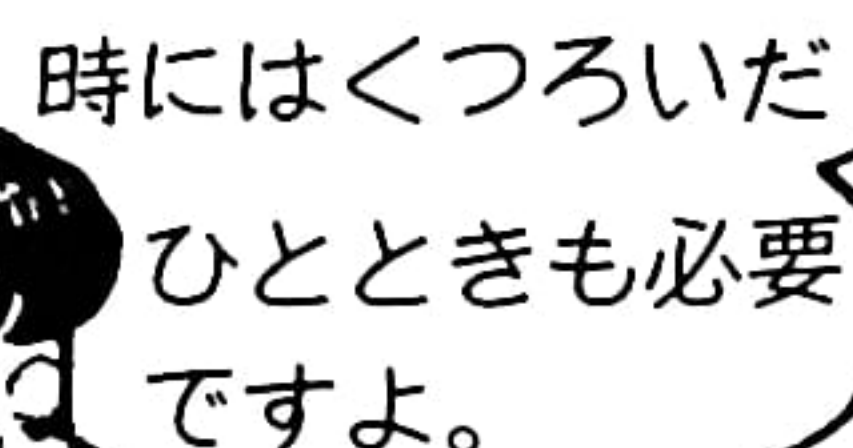
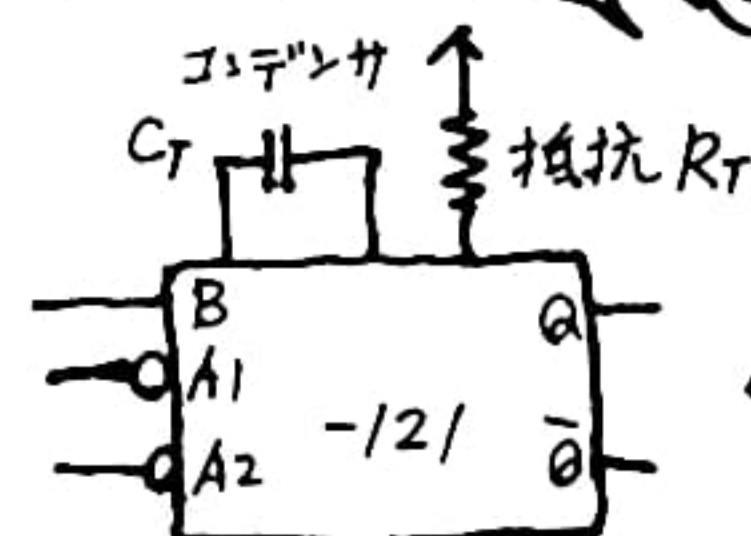
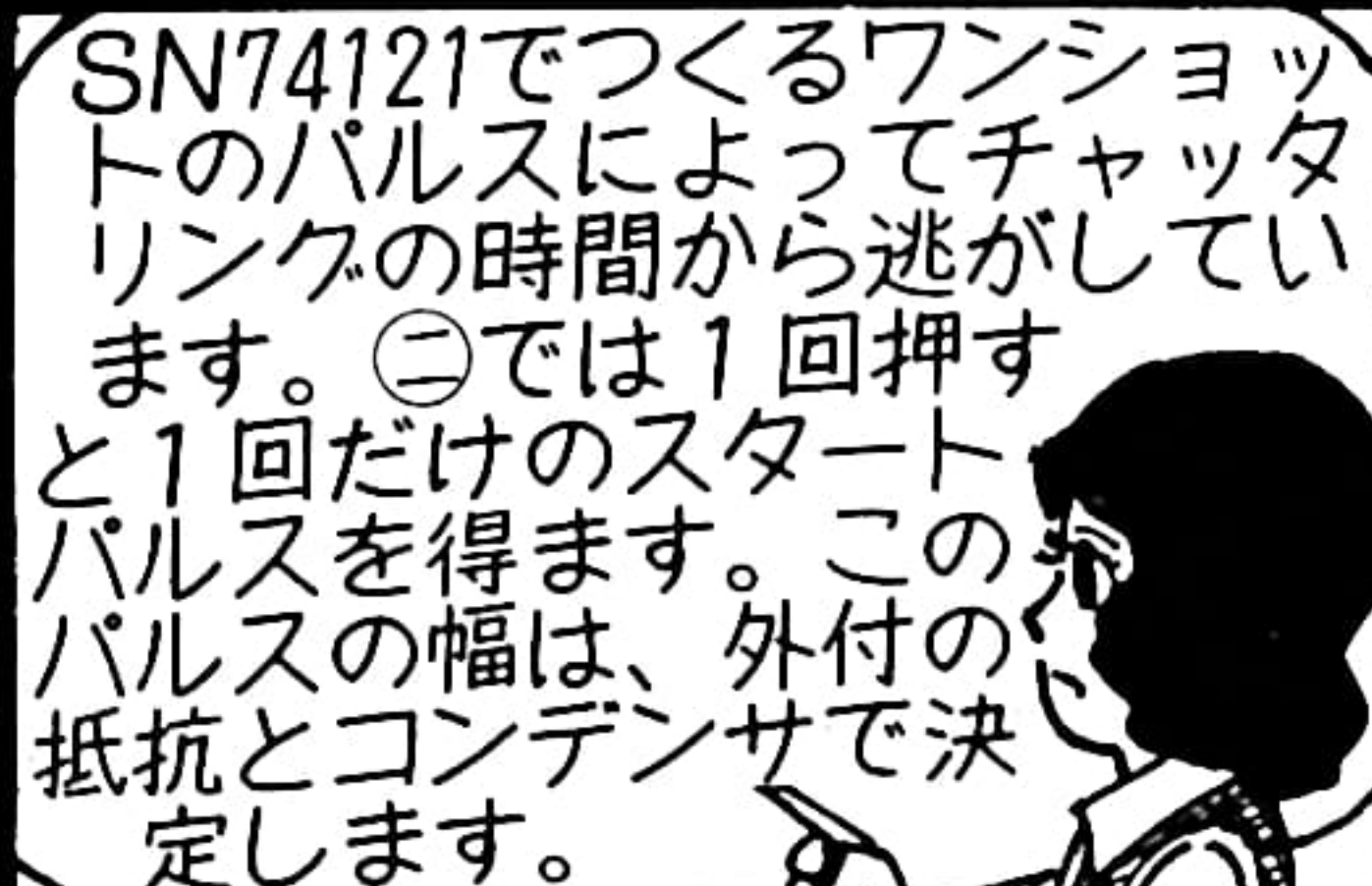
単位は便利なようにC_TをμF、RをKΩにすればT_pはmsecと、なります。

例1. $C_T=10\mu F$
$R_T=10K\Omega$ とすれば
$=0.7\times10\times10=70$ msec

1ms（ミリ秒）は0.001s（秒）です。

SN74121でつくるワンショットのパルスによってチャッタリングの時間から逃がしています。④では1回押すと1回だけのスタートパルスを得ます。このパルスの幅は、外付の抵抗とコンデンサで決定します。

コンデンサ C_T 抵抗R_T B A1 A2 －121 Q Q̄

時にはくつろいだひとときも必要ですよ。

フリップフロップやワンショットパルサーはマイコン出現まではIC回路になくてはならないものでしたが…

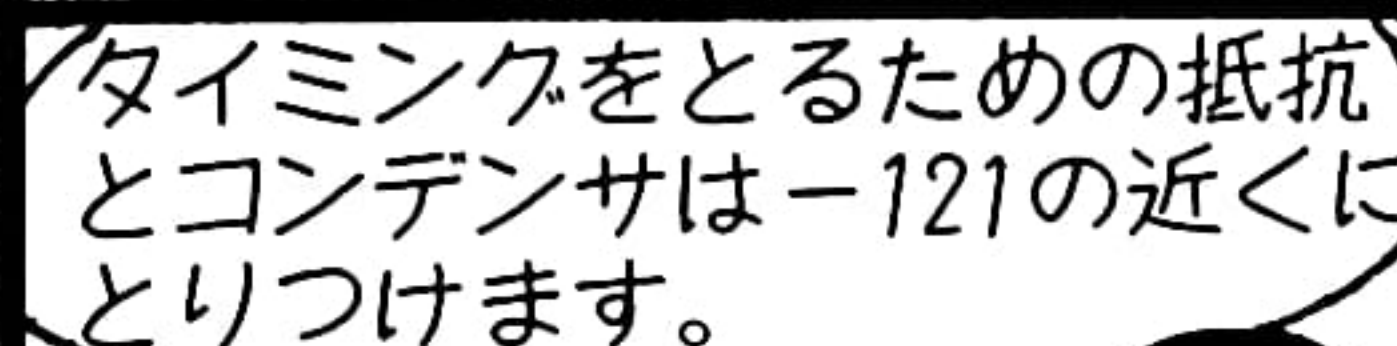
チ－ン チ－ン

タイミングをとるための抵抗とコンデンサは－121の近くにとりつけます。

10μF 10KΩ 74121

約70msのパルスがつくれます。

⑤ROMとRAM

ロムちゃんとラムちゃんはこれまで何度となく登場しましたがハード的なことをもう少しお話しましょう。

まず、ROMちゃんです。ROMちゃんはデータの読み出し専用メモリです。ということは、あらかじめ何らかの方法でデータが書き込まれていなければなりませんね。 その書き込み方から見て３種のROMがあります。

ROM
マスクROM
P-ROM
EP-ROM

マスクROM、これは量産製品に組み込む場合に適しています。ROM製造過程でプログラムされているので改めて書き込む手間が省けます。そのかわり他の製品に使用することはできません。

誕生即成人式

P-ROMは、プログラムできるROMの略です。これはユーザーが自分のプログラムを書き込み機を使ってデータを入れられるということです。RAMへの書き込みとは次元が違うものです。

RAM
ICと電圧レベルで書込める。
P-ROM
専用書込機を使って書込む。
異質のもの

そのプログラムも回路の一部を破壊することによって作るので一回書き込んだら変更はできないのです。失敗したらそのROM１個オシャカということになります。

08

そこで失敗したり仕様変更になってオシャカになるはずのROMをもう一度よみがえらせるようにしたのが、３つ目のEP-ROMというものです。Eはイレイザブルつまり書き込んだデータを消すことができ、元の状態に戻せるものです。

紫外線
EP-ROM

紫外線を照射することによって消すものが一般的でしたが、このごろは電気的に消せるものもあるようです。このEP-ROMの出現によってROMのムダ使いがなくなりました。

多品種特注品向き！

ROMを書き込み方法で分けると……
☆マスクROM…製造過程でプログラム量産品向き
☆P-ROM……ユーザーがプログラム。一度書き込むと修正できない
☆EP-ROM……ユーザーがプログラム。消去用エネルギーを加えれば再度プログラム可能
この３種類になります。

ROMちゃんのデータは、データバスに対して出力として働くのでセレクトされない時はバスに影響を与えないものでなくてはなりません。

データバス
ゲート
D7 D6 D5 D4 D3 D2 D1 D0
出力コントロール

そのため、ROMのデータ側はオープンコレクタ、またはトライステート・ゲートになっています。

出力 入力
オープンコレクター
ゲート
出力 入力
トライステート・ゲート

オープンコレクタのものは応答速度の関係で今ではあまり使えないようです。トライステート・ゲートについてはもうすぐ出てきます。次はRAMちゃんです。

CPU
ROM RAM RAM

RAMちゃんはメモリ保持の方法で次の2つに分けられます。ひとつはダイナミックRAMで、他のひとつはスタティックRAMです。

RAM

スタティック

ダイナミック

ダイナミックRAMは記憶素子がコンデンサなので一定時間ごとにデータを読み出して再び書き込むというリフレッシュ回路が、心要なのです。

リフレッシュ回路

ダイナミック

リフレッシュというわずらわしさはありますが、大容量のメモリが安価にできるという利点がありますので、パーソナルコンピュータに向いています。

RAM 64 Kバイト

一方スタティックRAMはトランジスタで双安定を保つため、リフレッシュという回路のわずらわしさはありません。ただメモリのバイト当りの単価が高いのです。4Kバイト以下のマイコンに多く使われています。

制御用とか、ちょっとしたデータ処理付秤量機なんかは、スタティックRAMを使っています。

×○用制御盤

8080Aや8085AはスタティックRAMしか使えませんが、Z-80にはリフレッシュ用のカウンターがついています。

それから、ダイナミックRAMのアクセス時間は10nsecぐらいで、スタティックRAMとは1桁違います。

データでROMちゃんと違うのはRAMちゃんのデータは双方向性だということです。データバスの信号の方向がCPUからのコントロールによって変わります。

データバスは8ビットです。1ビットは1本の導線で伝達でき特殊なものではありません。

CPU

スタティックRAM

D0 D1 D2 D3 D4 D5 D6 D7

回路図をよく見ると、RAMのデータピンはIN、OUT2種のものがあります。1種のものはパッケージの中の回路でつないであるのです。

データバス

RAM

IN

OUT

この中はどうなっているのかというと、トライステート・ゲートがIN、OUT用にひとつずつあってRead/Write信号でコントロールしているのです。

WRITE ゲート

DI

DO

READ ゲート

RAM 記憶回路

同時にゲートをあけるようなことをしなければ出力側を並列につないでも回路が混乱しないようにできているのがトライステート・ゲートです。

G

1 ....電流が外部に出る。

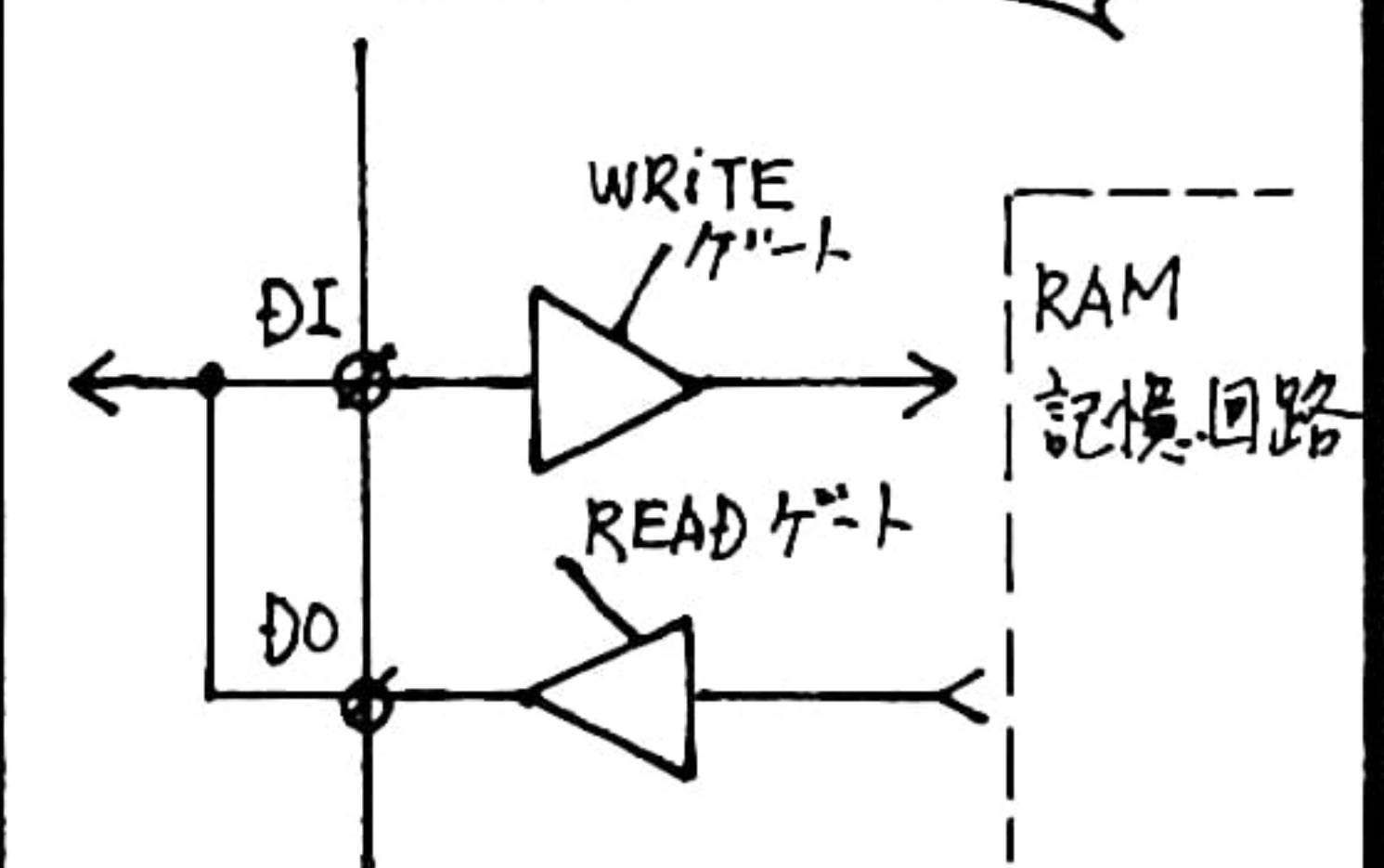
0 ....電流が外部から入る。

HiZ ...電流が流れない。(ハイ・インピーダンス)

それからRAMちゃんの場合はとくに読み出し、書き込みにそれぞれ時間がかかるということを考えておかねばなりません。では、これでRAM、ROMメモリの説明はおしまいです。

書き込み Write Pulse .... 250 nsec Min

読み出し Access Time .... 450 nsec Max

スタティックRAM 8101A-4の場合

マイコンで8ビットCPUといっているのは、データとして1度に8ビット読み書きができるCPUということなのです。

16進表現 A 7

1010011

CPU

データ

RAM

⑥メモリ容量とチップセレクト

ROM

RAM

RAM

?

IC飼い

その読み書きするデータがどこにあるかはっきりさせるのがアドレスです。8ビットCPUでは16ビットのアドレス出力があります。

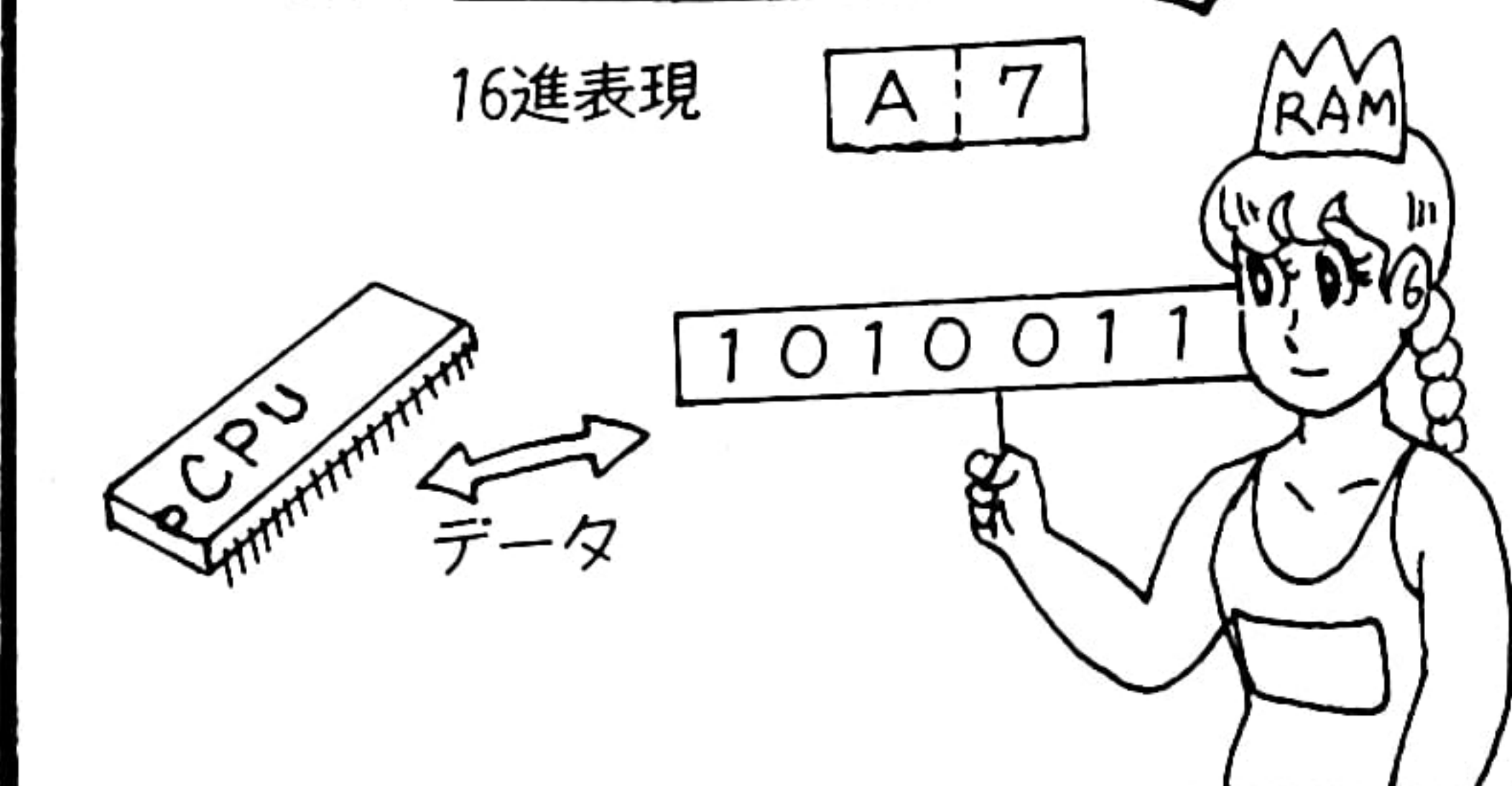
CPU

アドレス

0010010110011111

6 5 9 F

659F

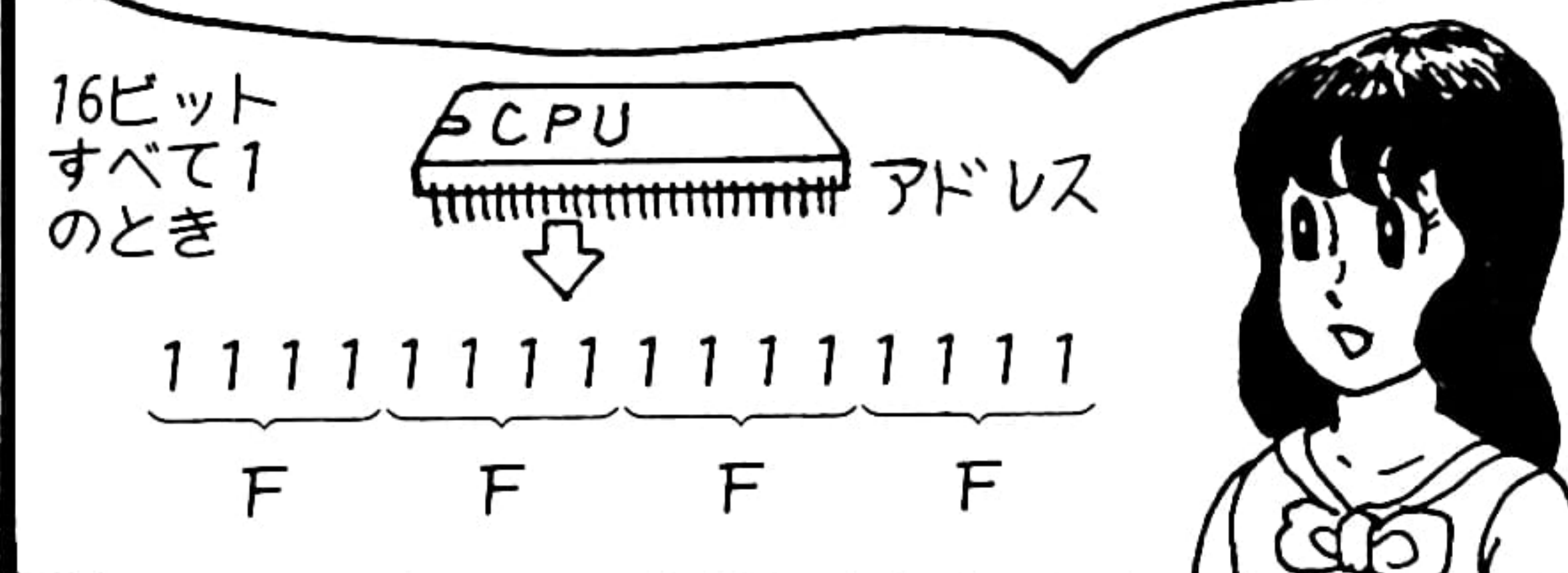
RAM

この16ビットで指定できる数は0000～FFFF までで10進数では65536通りです。普通パソコンのメモリは最大65K(キロ)バイトといわれるのはここからきています。

16ビットすべて1のとき

CPU

アドレス

1111111111111111

F F F F

このアドレスは製作時点で決まってしまいます。たとえばPC-8001 のメモリアドレスマップはこのようになっています。

CRT

PC-8001

ROM		RAM		
BASICインタプリタ と モニタプログラム	アキ	ユーザープログラムエリア	CRT表示用エリア	WORKエリア

0000 5FFF C000 F300 FFB7 FFFF

ところで使っているメモリはROMやRAMの種類があり同じRAMでも何個か組み合わせなければならない場合があります。

CPU

アドレス指定

メモリ

どのチップにご用ですか?

ROM

ROM

RAM

RAM

RAM

メモリはIC回路でできていて、1個のICをチップといっています。

ICのチップ

実際にはこんなバイト数のメモリはありませんが、今8バイトのメモリがあるとします。ROM、RAMはこの場合関係ありません。

0番
1
2
3
4
5
6
7

このひとハコは1バイト

私たちがプログラムする時には考える必要のないことですが、何個かのチップをどうやってつなぐのか参考までにお話しましょう。

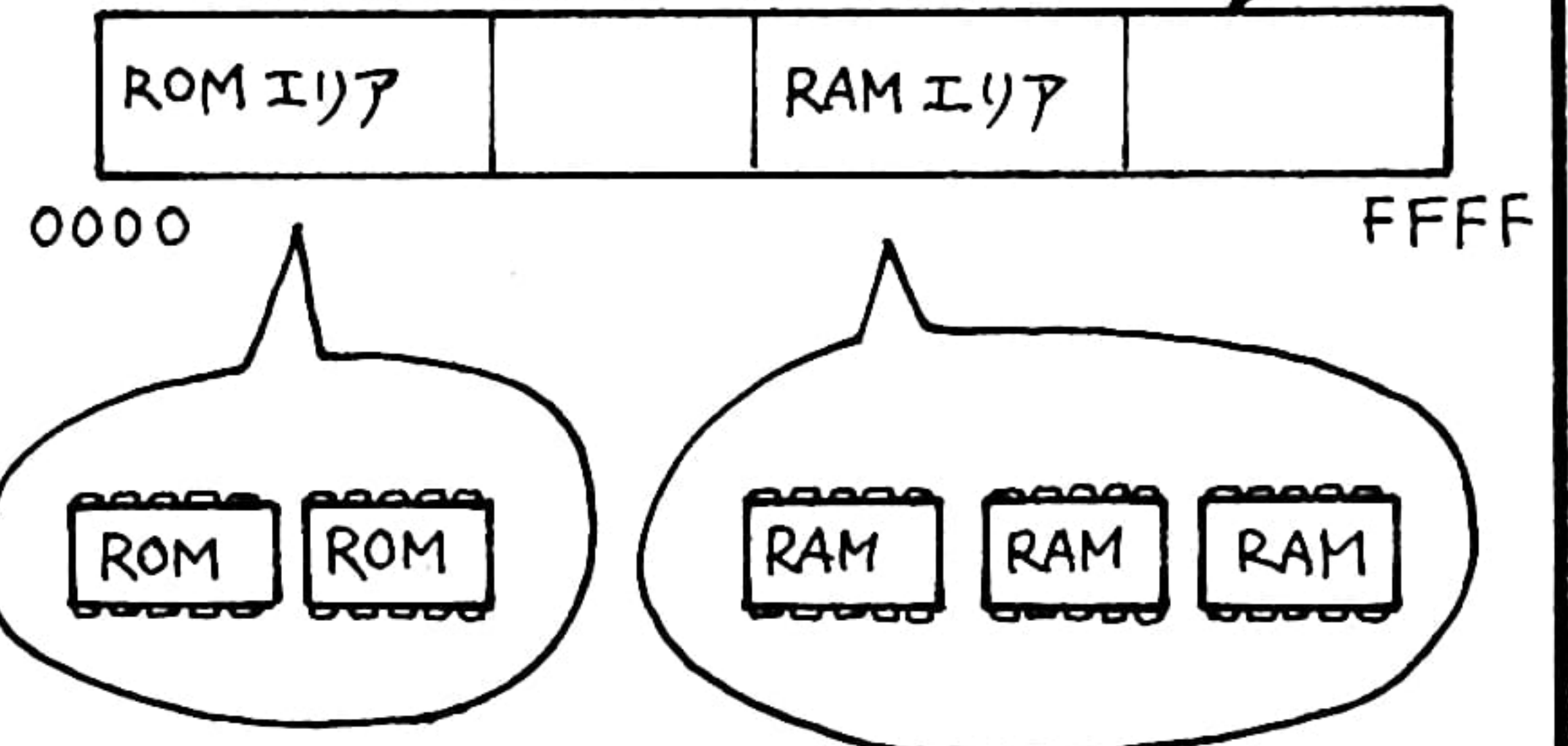
メモリマップは何個かのチップを組み合わせて作られるはず…。

CPUからみたメモリマップ

ROMエリア　RAMエリア

0000　FFFF

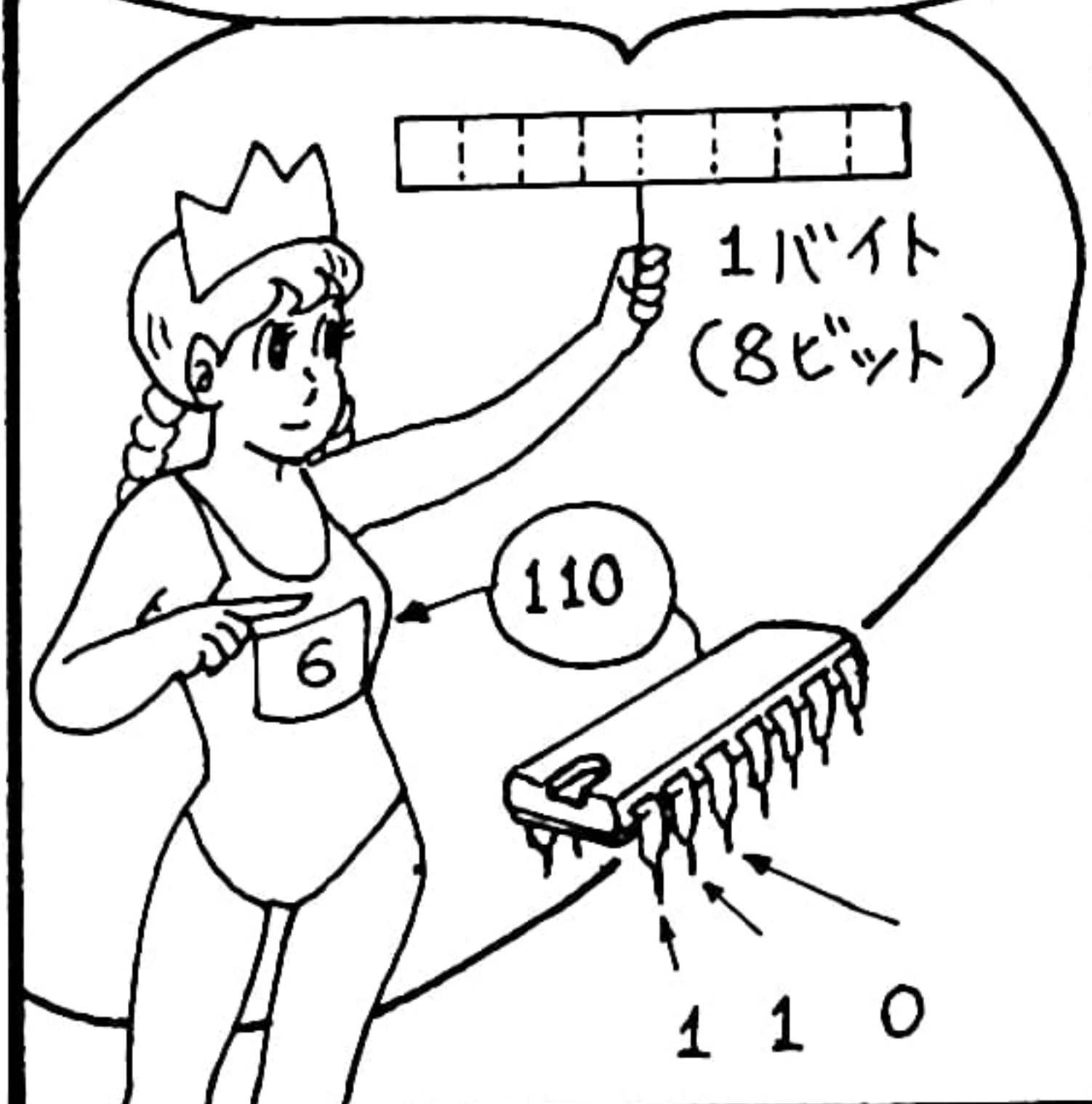
ROM　ROM

RAM　RAM　RAM

たとえば110という信号をアドレスピンに入れてやると6番のメモリを指定したことになります。

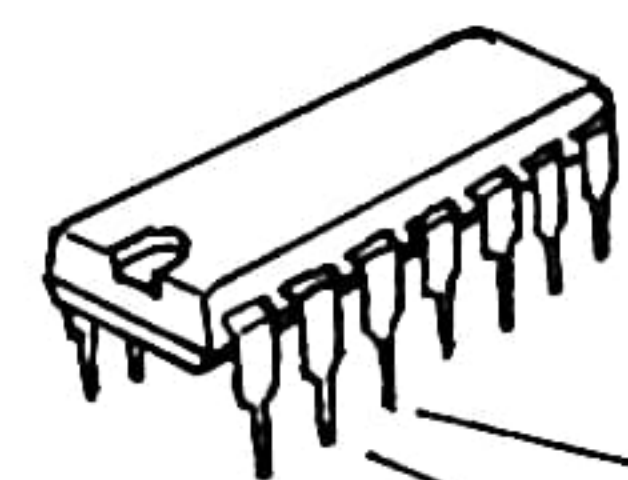
1バイト（8ビット）

110

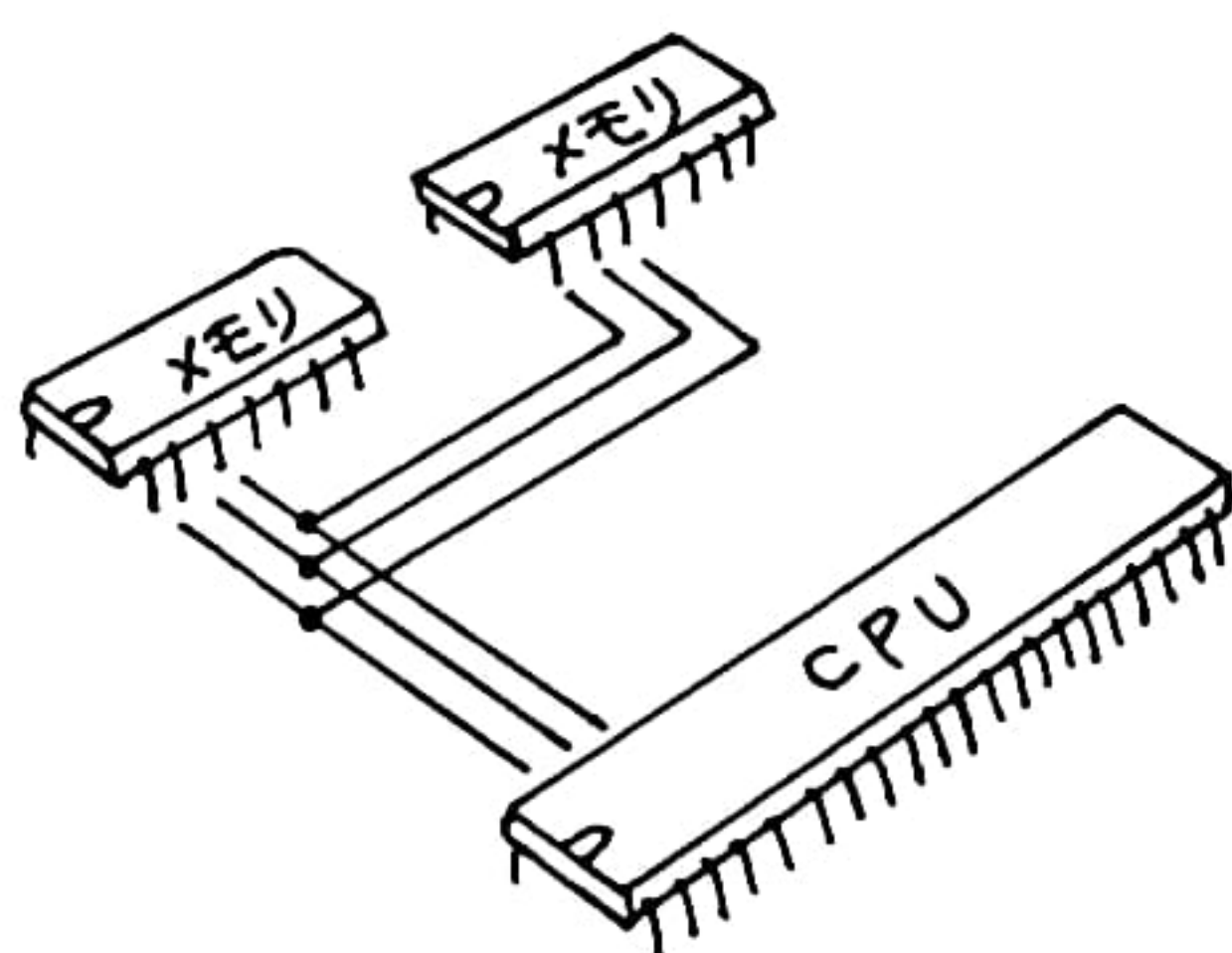
6

1 1 0

アドレスピンに入る信号と選ばれる1バイトのメモリの関係はこうなります。

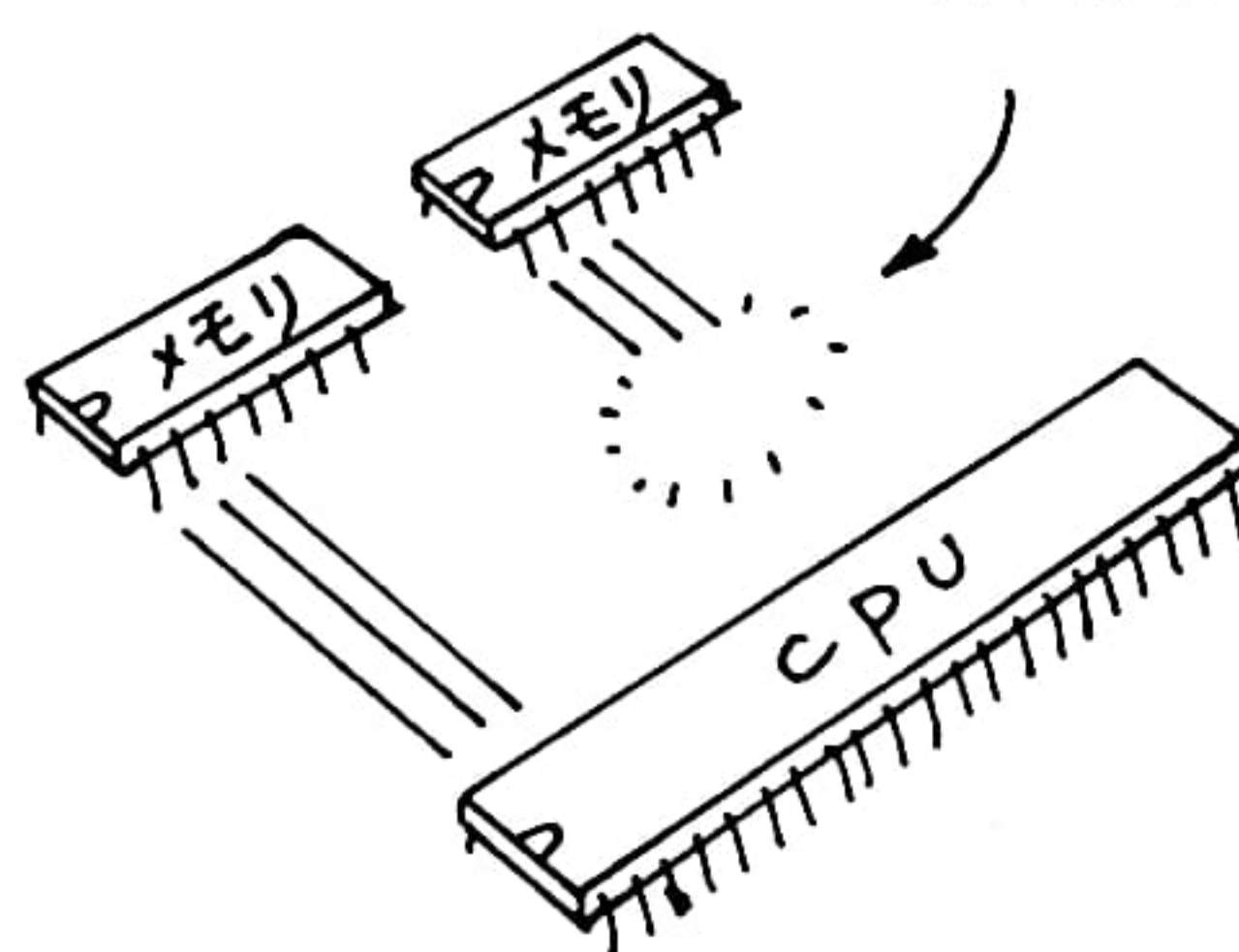
A2	A1	A0	選ばれるメモリ
0	0	0	0 番
0	0	1	1 番
0	1	0	2 番
0	1	1	3 番
1	0	0	4 番
1	0	1	5 番
1	1	0	6 番
1	1	1	7 番

この8バイトの中のどれかを指定するためには3ビットの信号があればいいんですね。ICのピンにはたぶんA0、A1、A2という3つのピンが出ているでしょう。

A2　A1　A0
アドレス指定用ピン

結論からいいますと、下のようにつないだらいいのです。ただ2つのチップを使いわける工夫が必要になります。

メモリ　メモリ　CPU

それでは同じメモリをもう1個使って0からFまで16バイトのメモリにしたい時はどうしたらよいのでしょうか？

どうつないだらよいか？

メモリ　メモリ　CPU

このアドレスピンをCPUのアドレスピンにつなげばCPUは0から7までの8バイトのメモリを持つことになります。

メモリ　CPU

このチップの使い分けをチップセレクトと呼んでいます。そうです。内部メモリにアドレスを持つICチップにはこのチップセレクトピンがついているのです。

メモリ
CS
チップセレクタ
A0 A1 A2

このチップセレクトピンで何番目のチップのメモリを使うのかを指定してやればいいのです。

メモリ
メモリ
0 1
選択回路

選択回路の先はCPUのもう1ビット上のA3につながります。これでうまくいくはずです。

メモリ
メモリ
0 1
選択回路
CPU A3A2A1 A0

CPUから出るアドレス信号で0〜7の場合はA3は0ですから0番のチップが選ばれています。

0番をセレクト
A3
0

次に8〜Fのアドレスが出力された場合はA3は1ですから1番のチップが選ばれるというしかけです。

1番をセレクト
A3
1

チップの数が増えたらA4、A5…と上位アドレスを選択回路につなげばよいわけですね。

‥7 6 5 4 3 2 1 0
D C B A
CPU A6 A5 A4 A3 A2 A1 A0 アドレスピン

この選択回路は入力された2進化コードを10進化して出力するものでいいですね。このような機能をもった I Cもちゃんと（マイコンより昔から）売られているのです。

2進→10進変換 I C

インプット				アウトプット				
D	C	B	A	0	1	2	3	4
0	0	0	0	L				
0	0	0	1		L			
0	0	1	0			L		
0	0	1	1				L	
1	1	1	1					

0番 8バイト
1番 8バイト
0000 0007 000F
メモリマップ

1番チップ 0番チップ
メモリ A0 A1 A2 CS
メモリ A0 A1 A2 CS
コード変換チップ
3210
CBA
A3 A0 A1 A2
CPU

これで一応0からFまでのメモリアドレスマップができました。

チップ5つ
つなぐには…

0番のチップ　1番のチップ　2番のチップ　3番のチップ　4番のチップ

A2 A1 A0 CS

CPU

アドレス出力
A0 1
A1 0
A2 1
A3 0
A4 1
A5 0

デコーダ
A B C
0 1 2 3 4 5

CPUは15というアドレスを出力しています。この時選ばれるチップは2番です。

コード信号を符号信号に変えるものをデコーダといいます。この場合入力が010で'2'のピンに出力があります。

アドレス出力
010101

そこでそれらを16ビットのアドレス0000~FFFFにうまく割り当てていくためには、このデコーダを用たチップセレクトが必要というわけです。

メモリは、ROM、RAMに限らず多くのチップから構成されます。

メモリの40番目は、
A5 A4 A3 A2 A1 A0
1 0 0 1 1 1
2 7
27(H)で終わります。
これは39(D)です。

37
38
39

メモリの1バイト目は
A5 A4 A3 A2 A1 A0
0 0 0 0 0 0
0 0
00(H)で始まり、

メモリ
00
01
02
03

上図を見てください。CPUのアドレスは、A0、A1、A2がメモリチップのバイトセレクトに使われA3、A4、A5がチップセレクトに使われています。

私たちが何気なく使うアドレスはハード的にはこのようにつながっていたのです。

CPU
アドレス出力

チップセレクト

バイトセレクト

実際のROMやRAMといったメモリは、もっと容量が大きいですよ。

8バイトメモリの5つのチップを使って00~27(H)番地までプログラムに使えるようになりました。

10進では0から39までの40バイトです。

⑦トライステート・ゲート

メモリやI/Oポートのデータ線はすべてCPUにつながっています。CPUのデータピンが8つですんでいるのはそれぞれのアドレスのデータがこの8つのピンを共有しているからです。

CPU

8ビットデータバス

第3の状態はハイ・インピーダンスでーす。

1（ハイ）

0（ロー）

そして……

HiZ

このあたりはCPU制御信号とも関係があります。じっくり読んでいってください。

だんだんむずかしくなってくるわ…。

HIZ

ハイインピーダンス

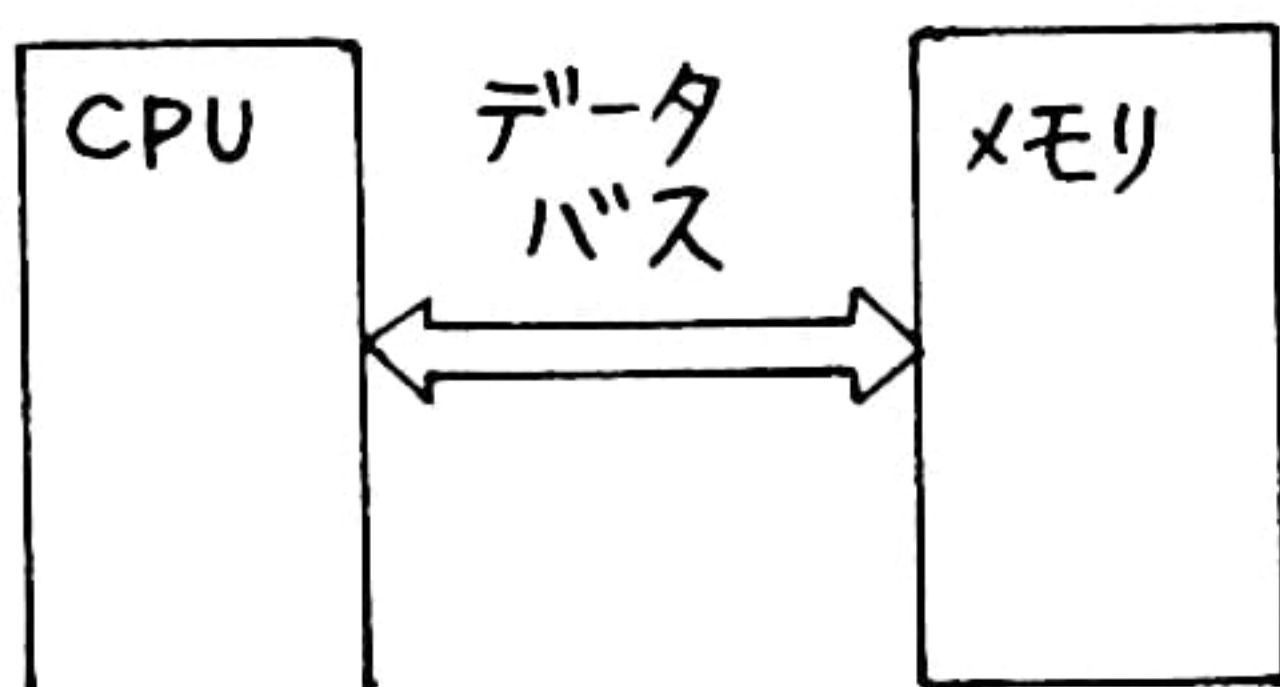
フローティング

浮いている

この矢印はデータの方向を示していますが、ある時間には←か→のどちらかにしか動きません。

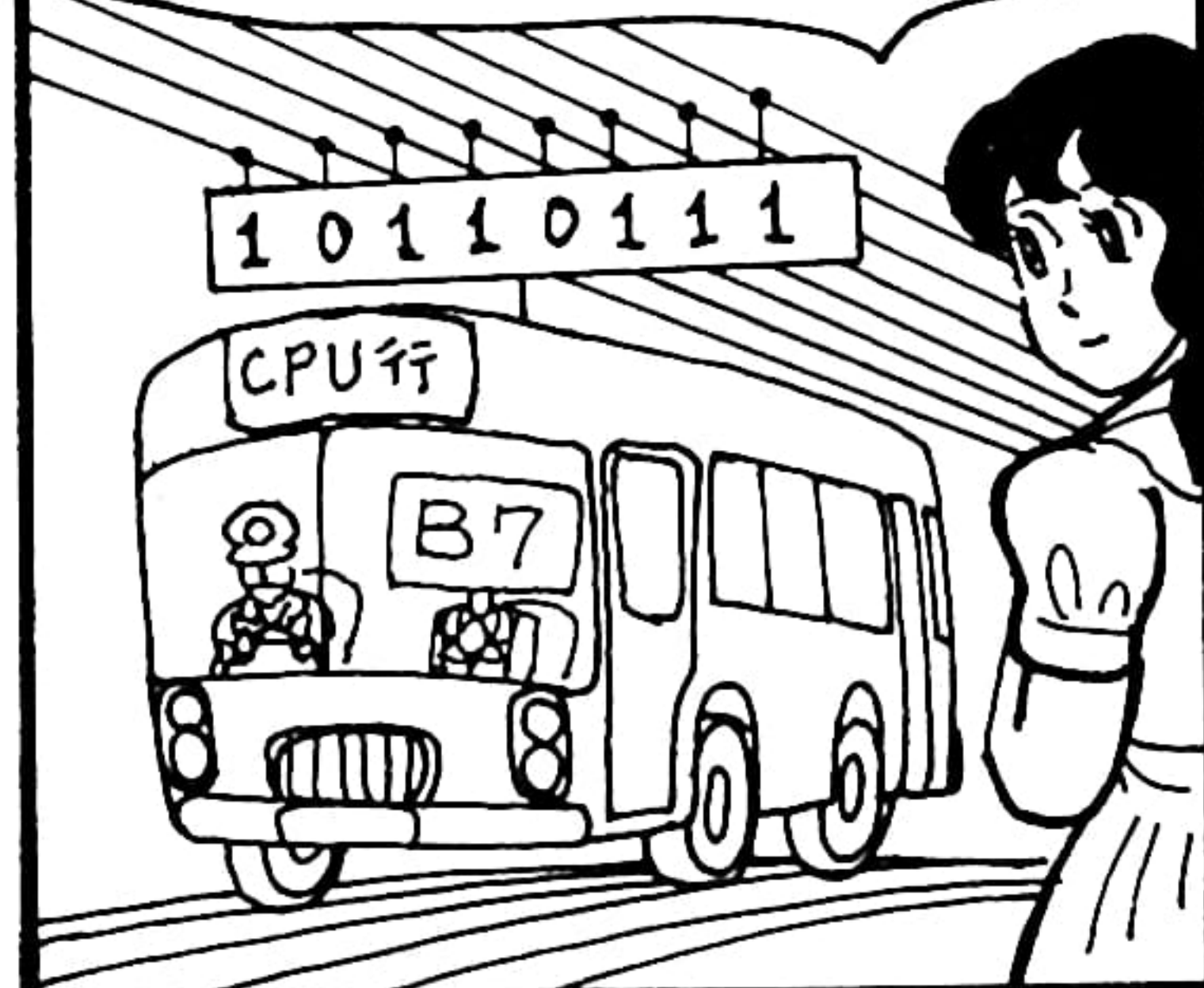
CPU

データバス

メモリ

このデータピンの共用がバスということになります。

10110111

CPU行

B7

下のトランジスタがONしている時は上のトランジスタはOFFとなり'L'を出力します。

L

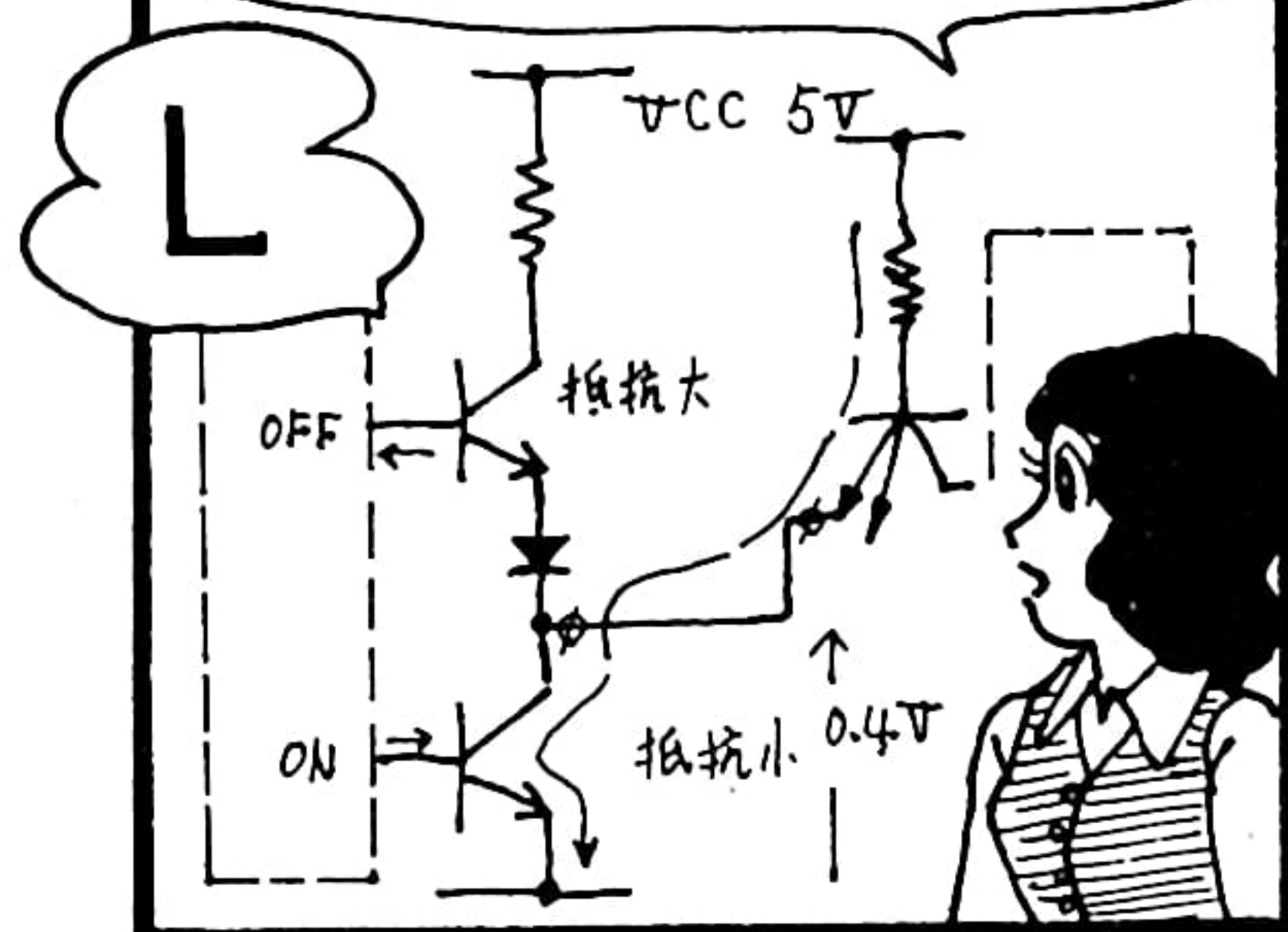
VCC 5V

OFF

抵抗大

ON

抵抗小

0.4V

上のトランジスタがONしている時は下のトランジスタはOFFとなり'H'を出力します。

H

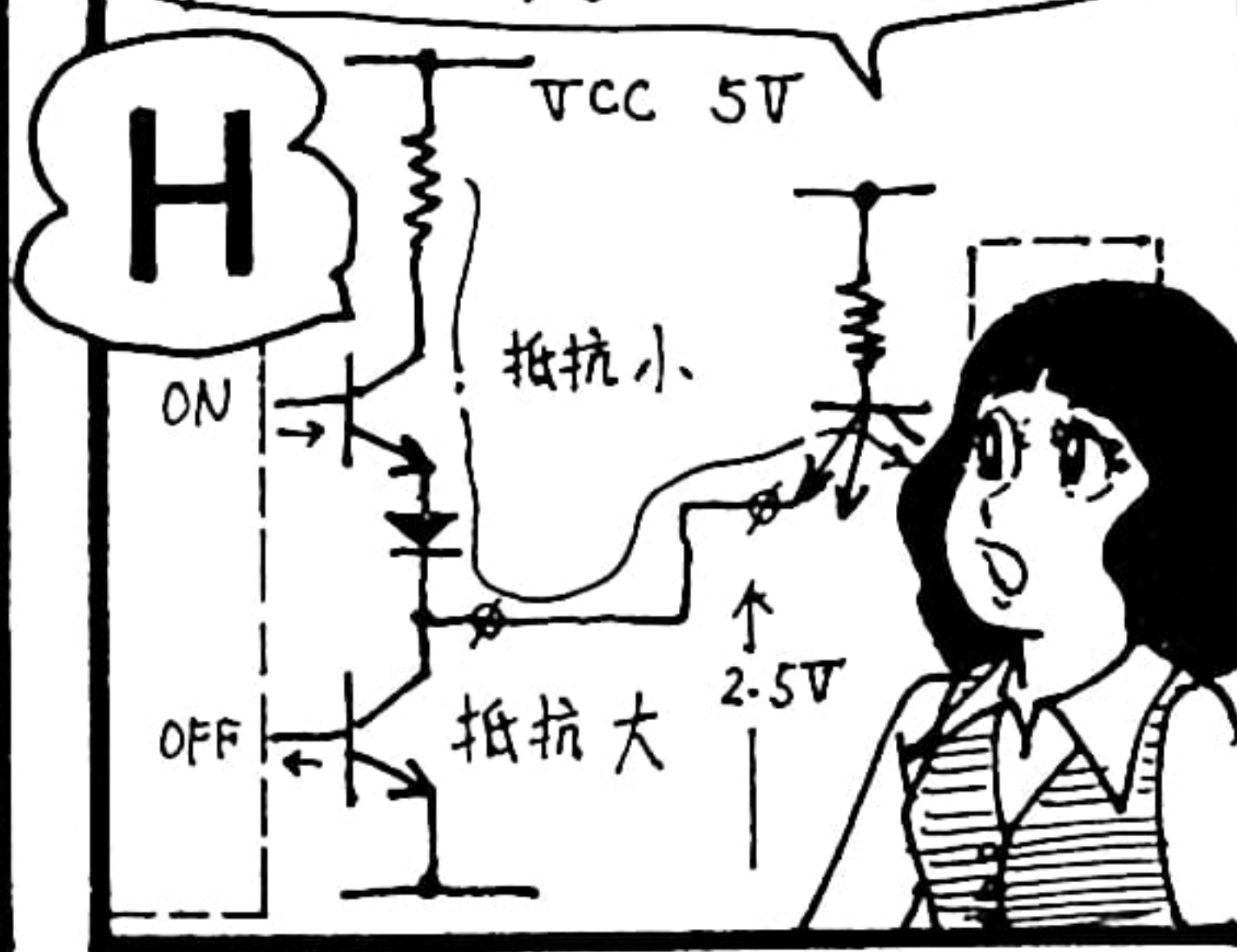
VCC 5V

ON

抵抗小

OFF

抵抗大

2.5V

普通のゲートはHかLのどちらかの出力以外の状態はとれません。

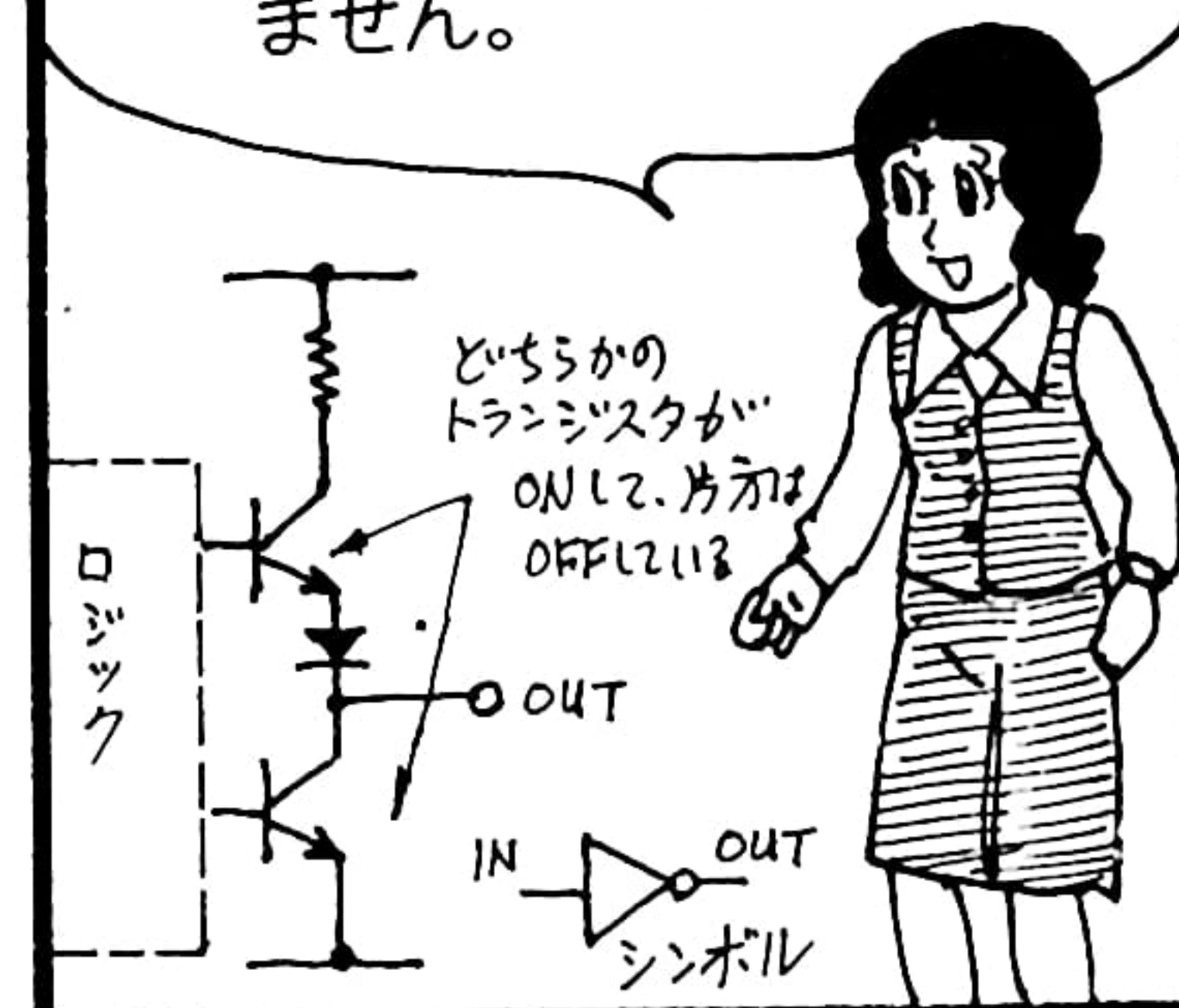
どちらかのトランジスタがONして、片方はOFFしている

ロジック

OUT

IN

OUT

シンボル

そこで上下のトランジスタを両方同時にOFFにするコントロール入力を設けたトライステートのゲートがあるのです。

B コントロール

A

C

入力

出力

IC2のTr4に電流が流れ込み、パンクする。

VCC 5V

IC1

130Ω

Tr1

小

大

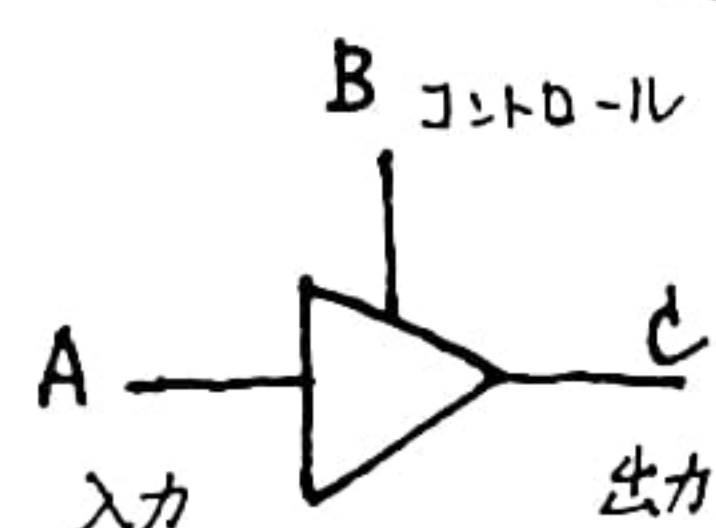
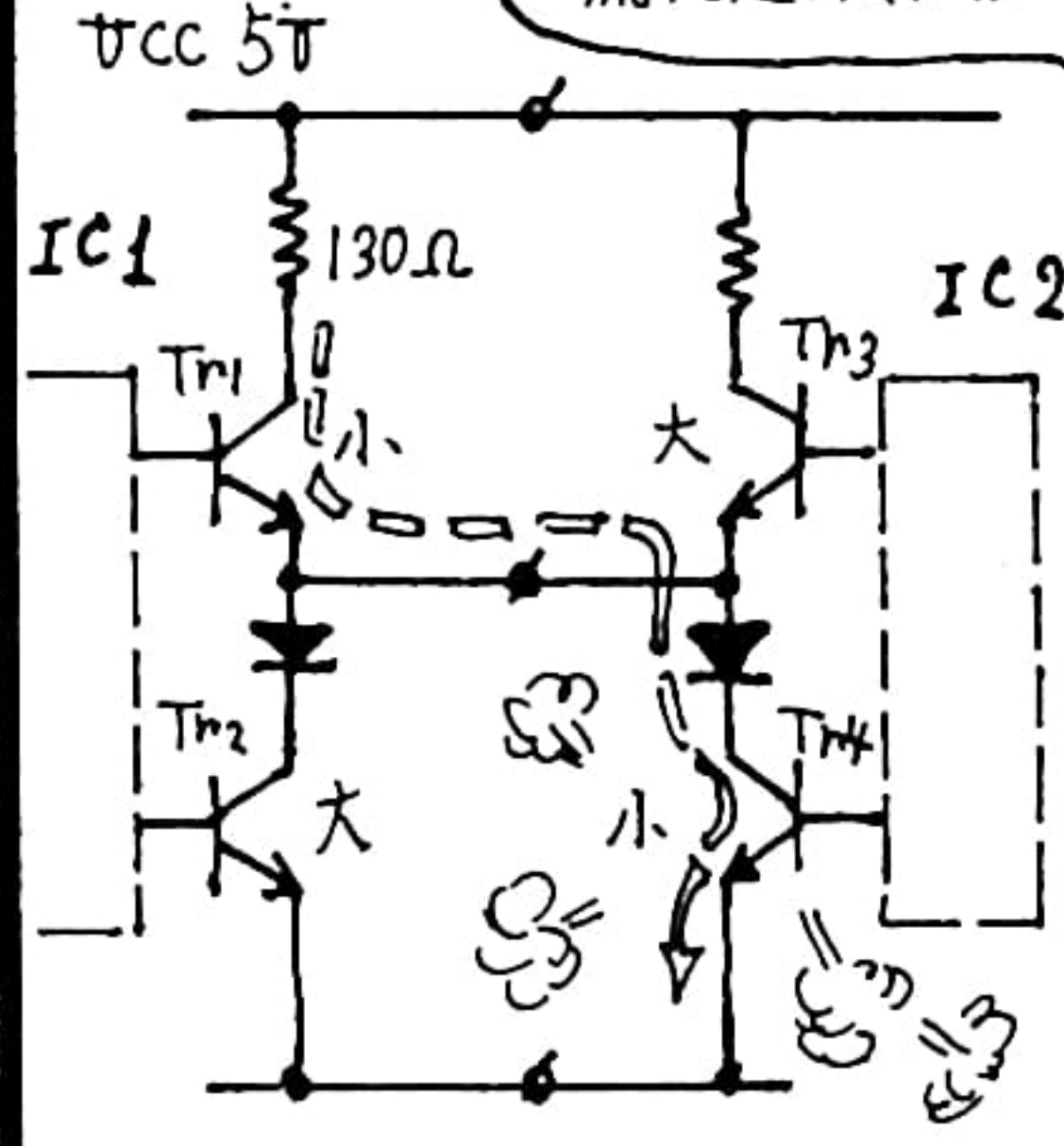
Tr3

IC2

Tr2

大

小

Tr4

右の▷▷の2つの出力部分です。

このようなトーテムポールの出力は並列に接続することができません。このような論理になっている時は、▷のゲートが焼けてしまいます。

L

H

H

L

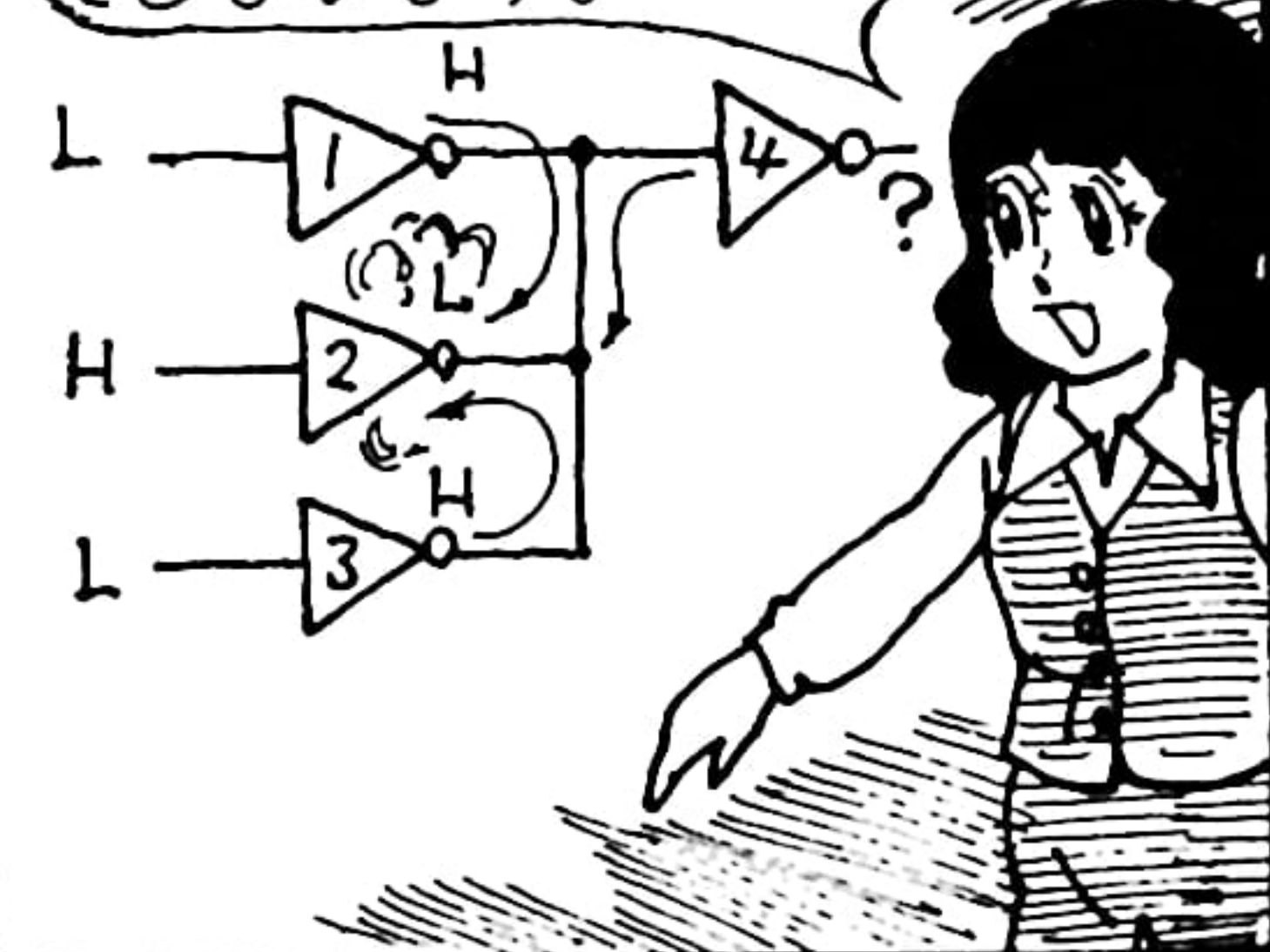
H

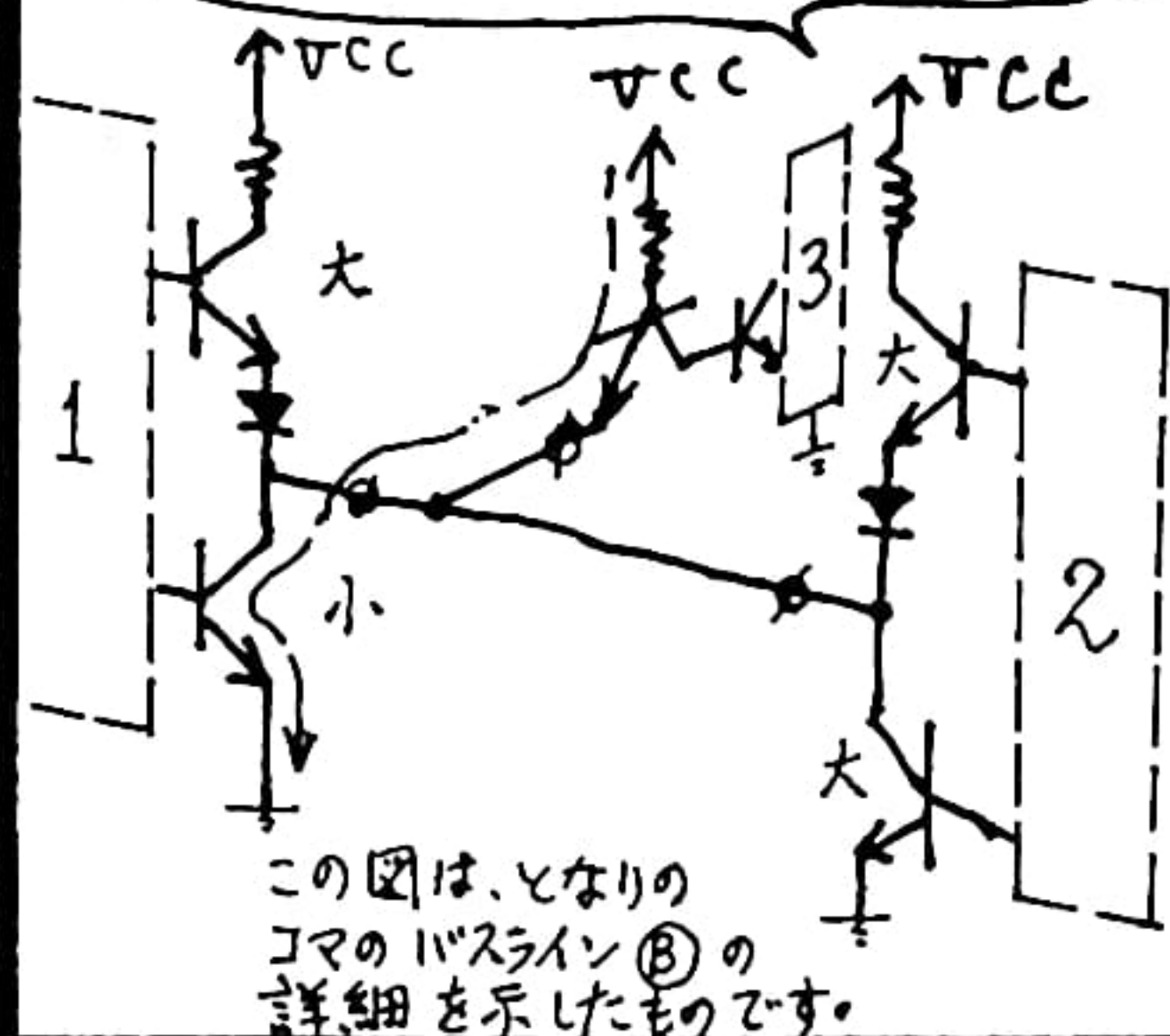
1

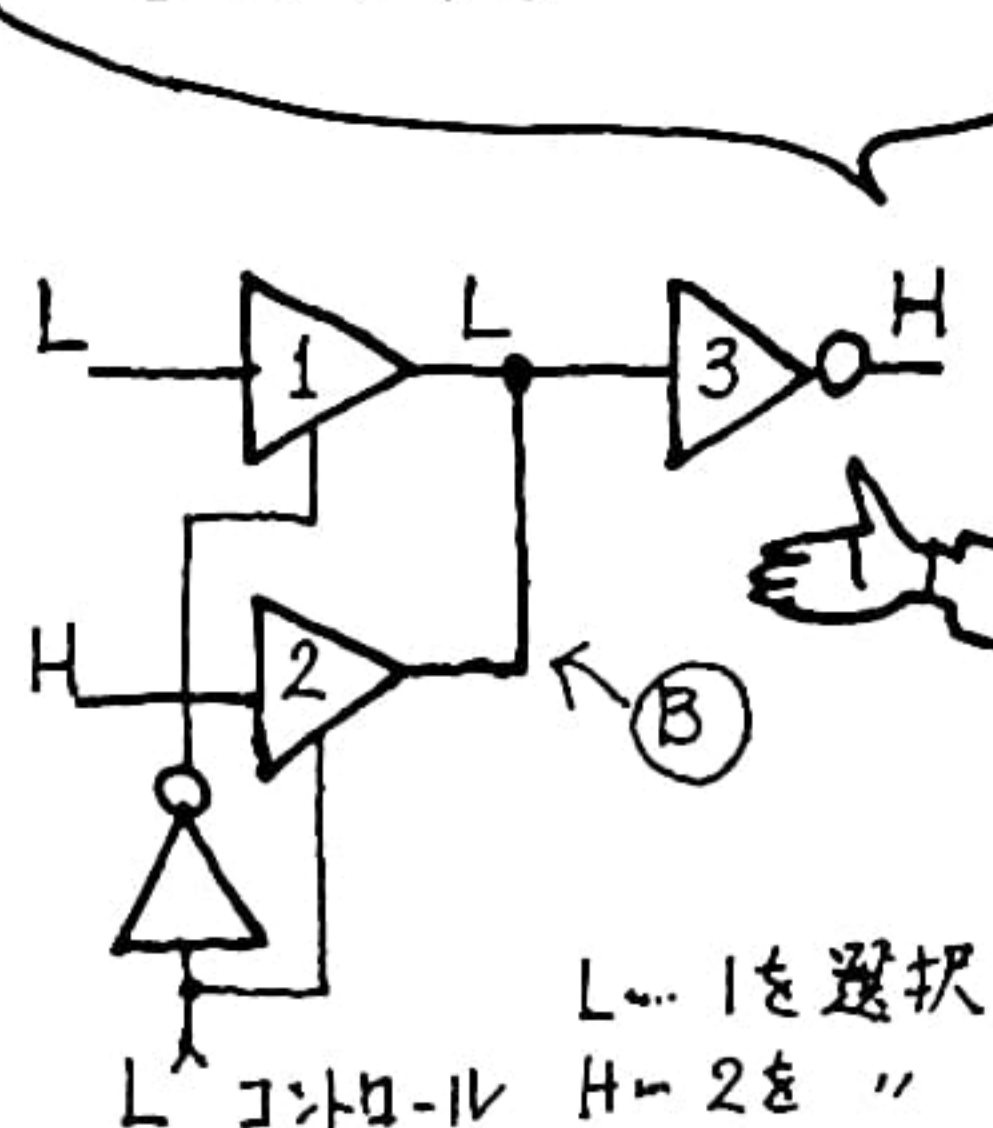
2

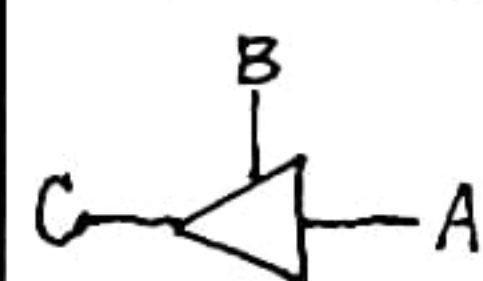
3

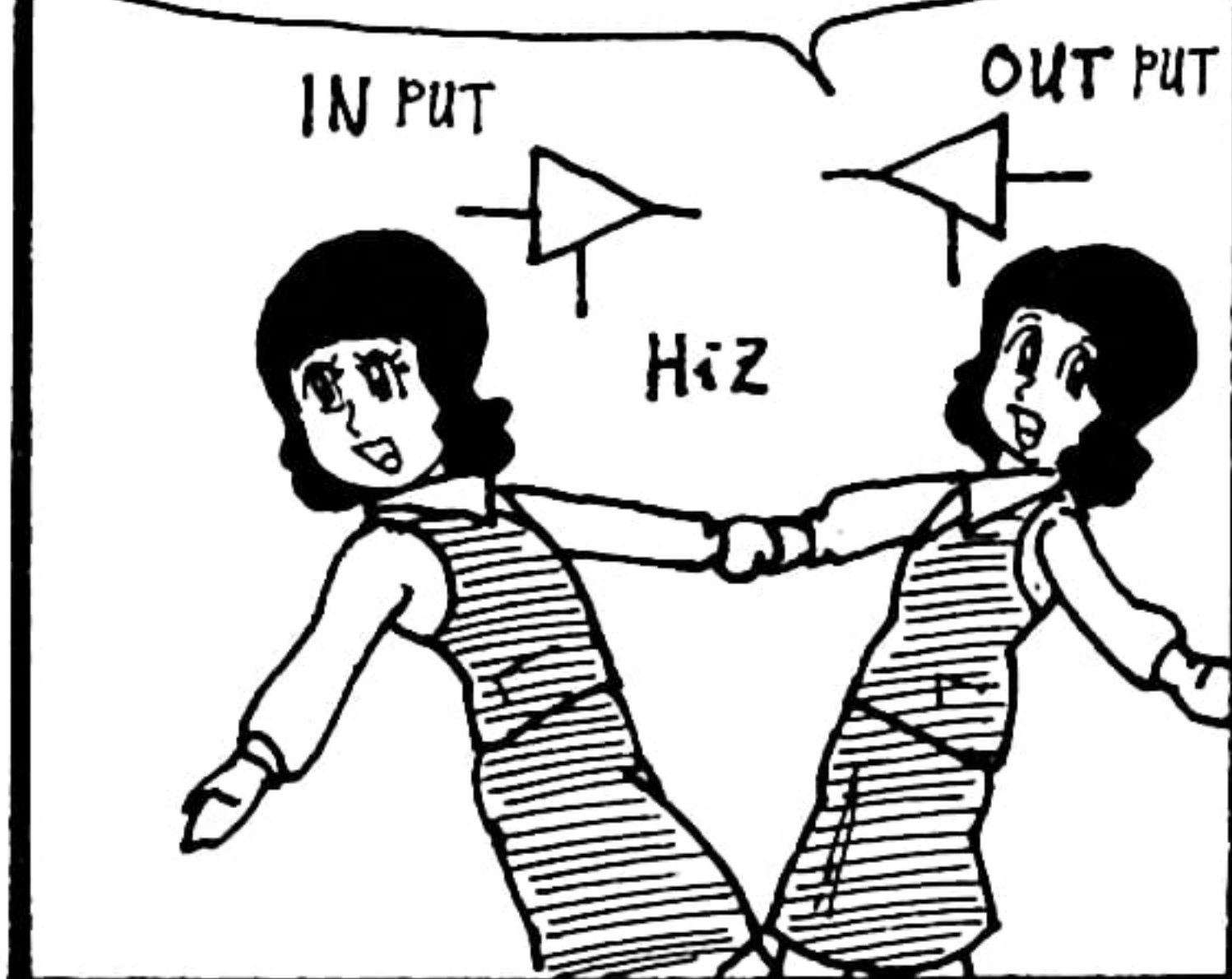
4

?

この場合IC2の出力はHiZとなり回路から切り離された状態です。この状態をフローティングというのです。

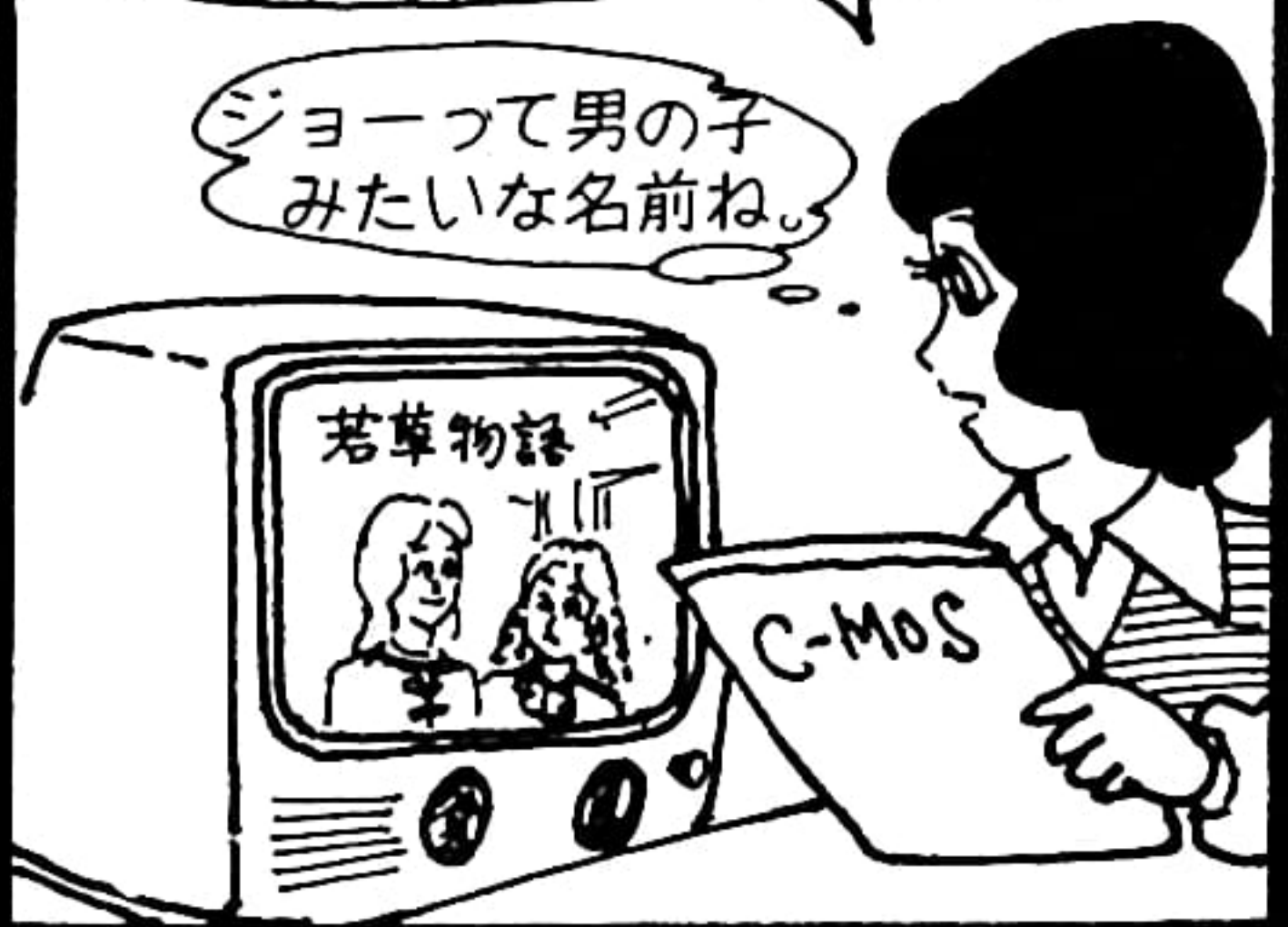
この図は、となりのコマのバスライン®の詳細を示したものです。

このように回路を組めば、出力同士のケンカはありません。だからバスラインとして使えるのです。

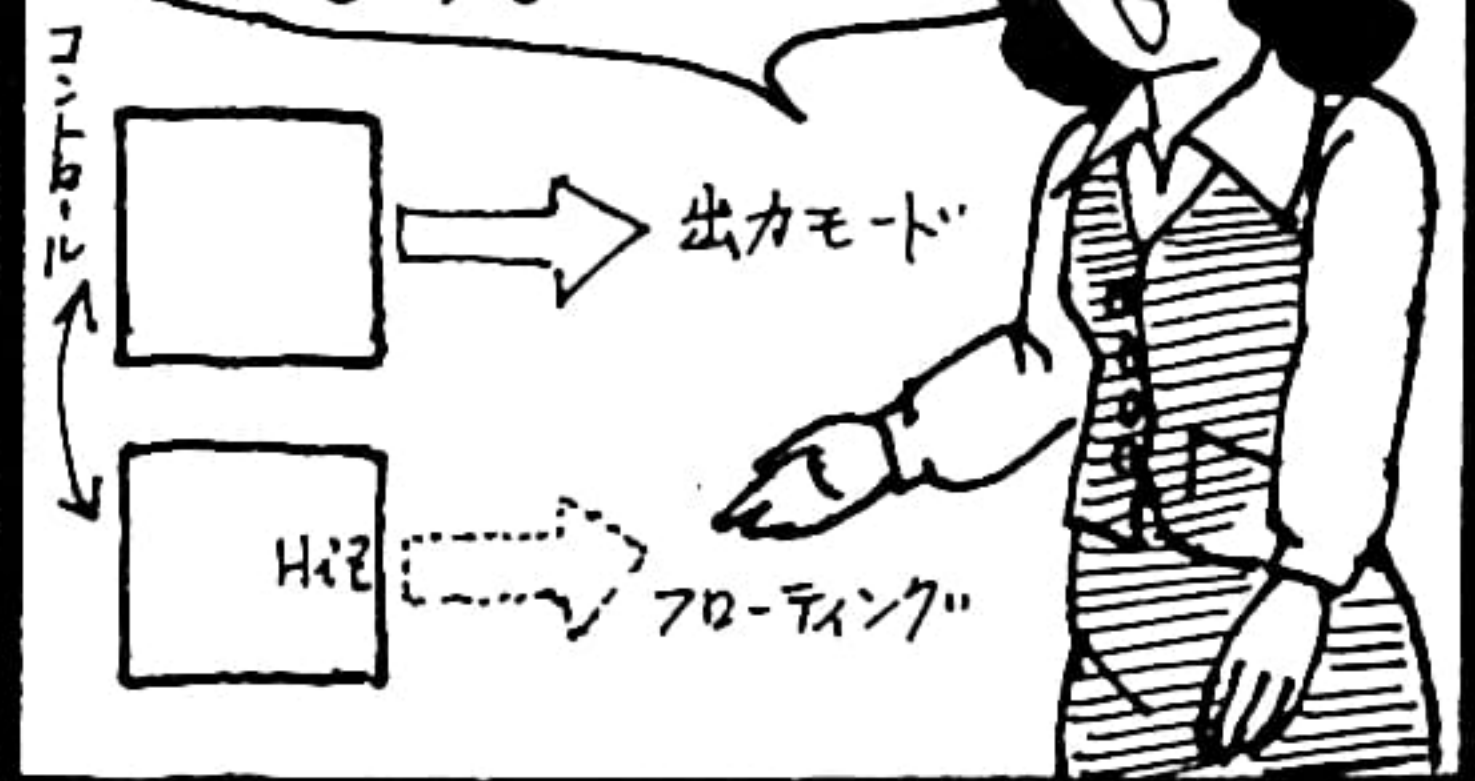
L…1を選択
H…2を 〃
コントロール

これが、トライステート・ゲートの真理値表です。HiZはハイ・インピーダンスのこと。上下のトランジスタがOFFの状態です。

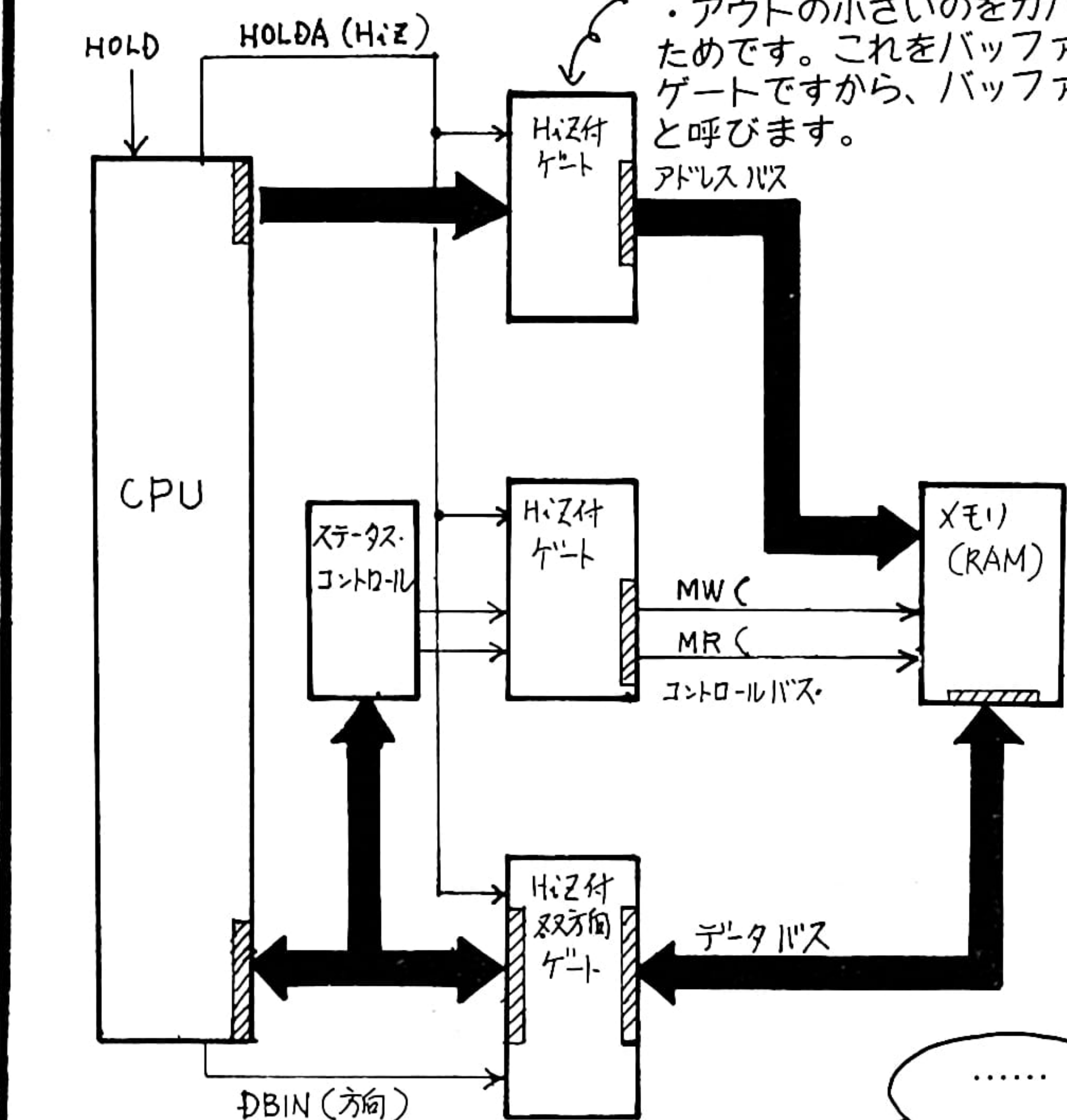
A	B	C
0	0	HiZ
1	0	HiZ
0	1	0
1	1	1

データバスの双方向性バスもこのトライステート・ゲートを応用しているのです。

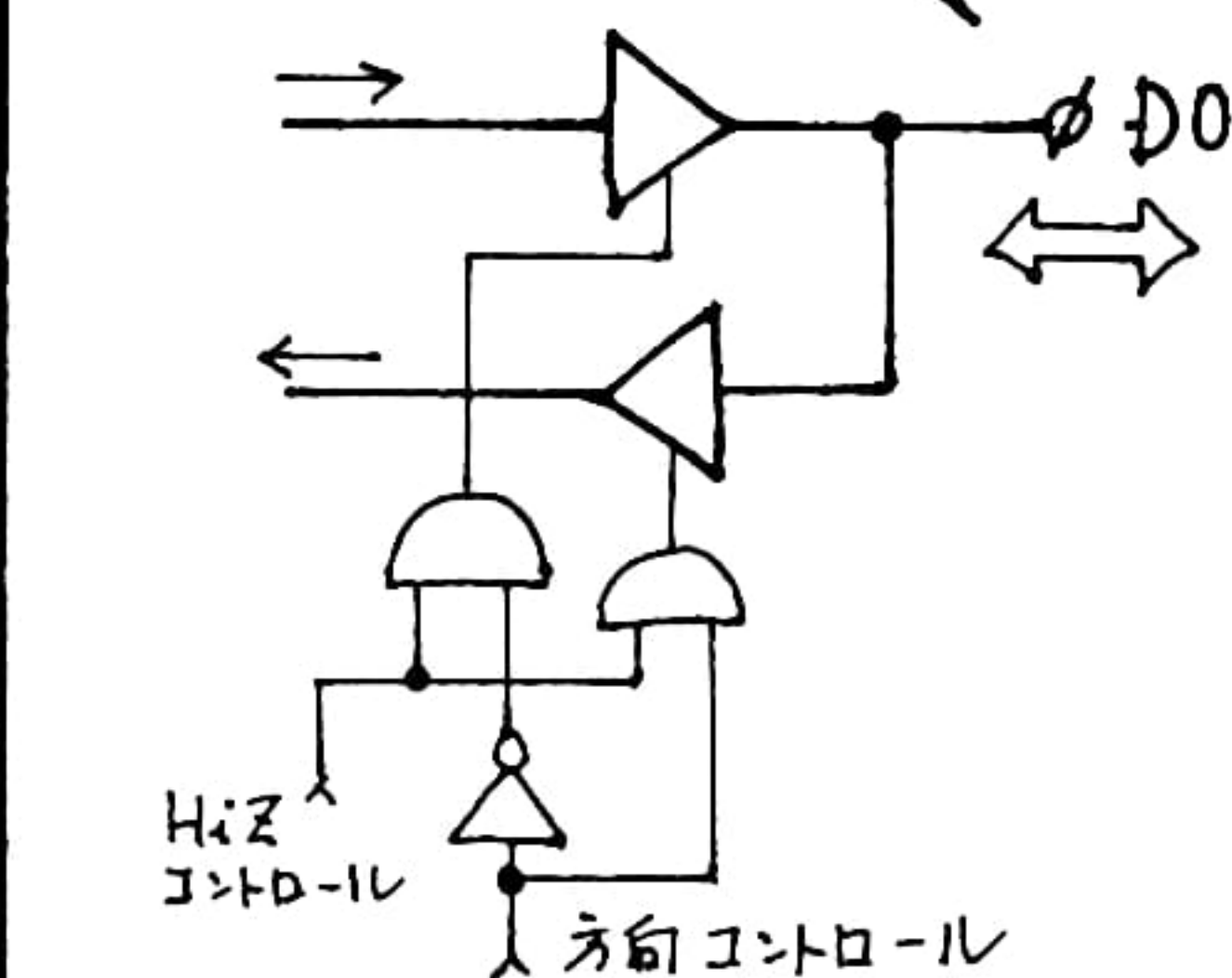
IN PUT
OUT PUT
HiZ

この回路はTTLのものですが、CPUやメモリはC-MOSです。でも、理屈は同じですから…。知りたければ、自分で勉強してくださる？ 私、ちょっと忙しいの。

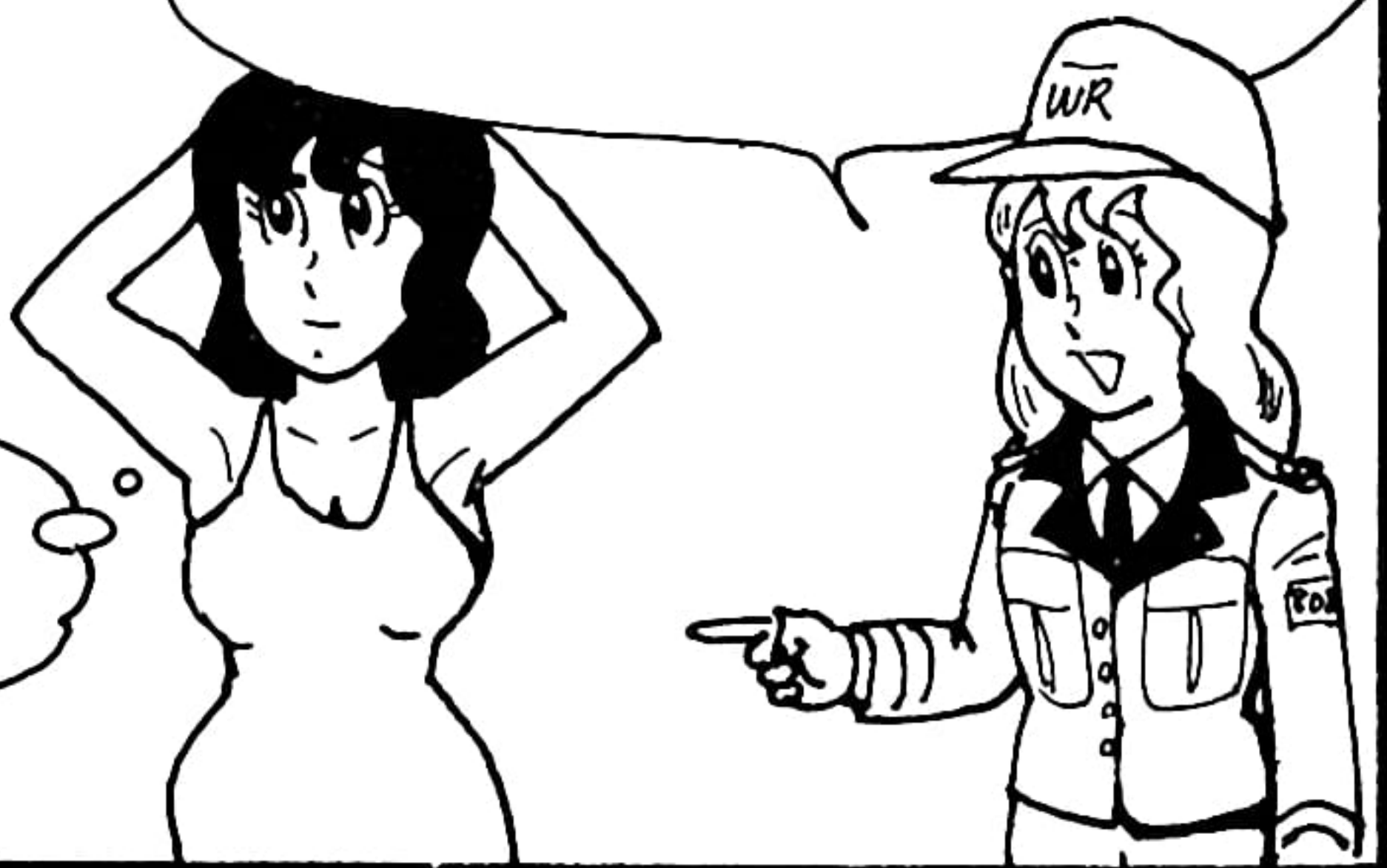
ジョーって男の子みたいな名前ね。

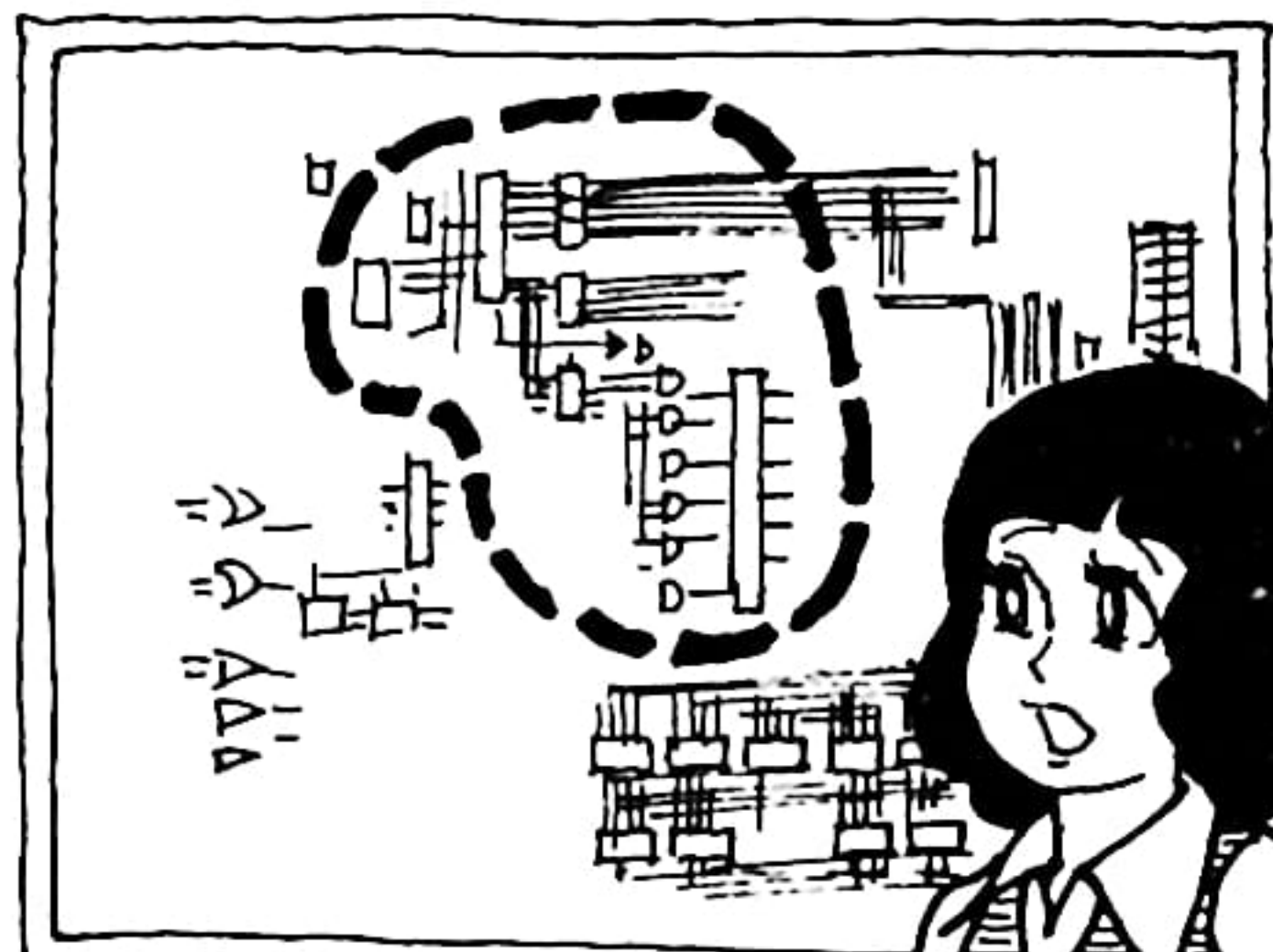
若草物語
C-MOS

バスラインに接続されているチップで出力端子側は、すべてこのトライステートになっています。そうでないと、出力同士のケンカが起こって混乱してしまいます。

コントロール
出力モード
HiZ
フローティング

3つのゲートは、CPUのファン・アウトの小さいのをカバーするためです。これをバッファといい、ゲートですから、バッファゲートと呼びます。

HOLD
HOLDA (HiZ)
CPU
HiZ付ゲート
アドレスバス
ステータス・コントロール
HiZ付ゲート
MW
MR
コントロールバス
メモリ (RAM)
HiZ付双方向ゲート
データバス
DBIN (方向)

このようにトライステート・ゲートを組み合わせたら端子DOはコントロール信号によって双方向性になりますね？

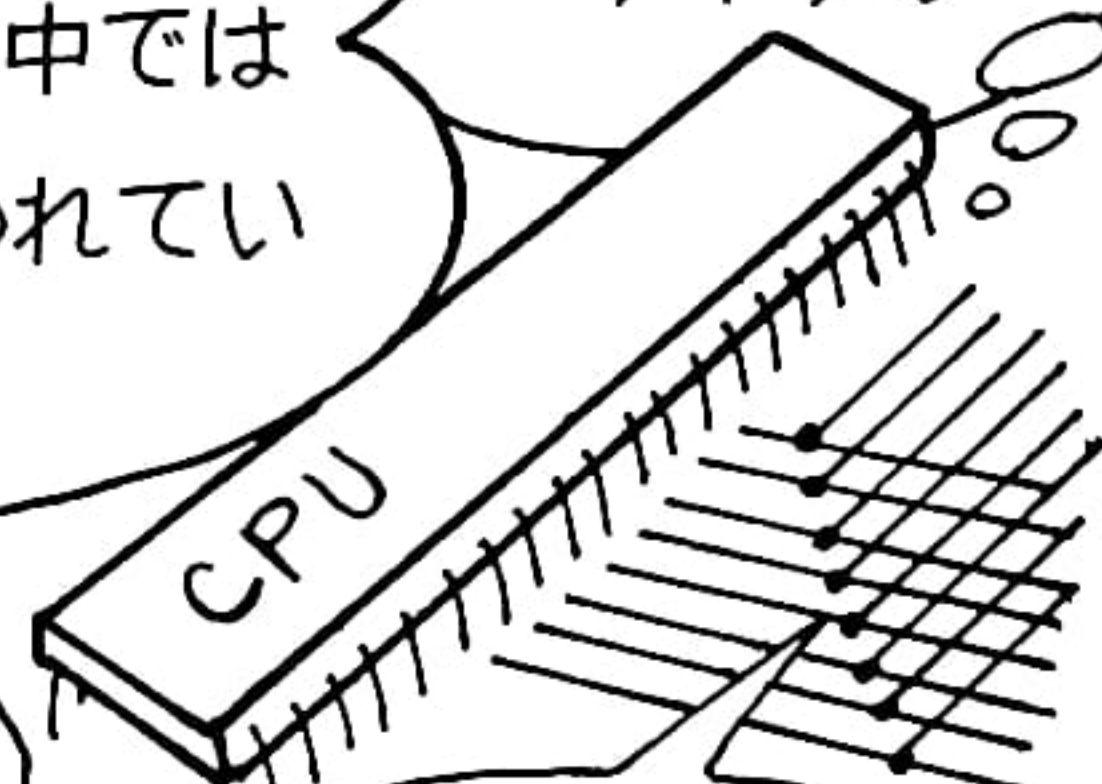
DO
HiZコントロール
方向コントロール

図中 ▭ が、ステートの出力部です。ですから選ばれていないアドレスの出力部はすべてバスに対して浮いています。

……
……。

⑧CPUと制御信号

これはCPU基板の図面ですが⬚内がCPUの領域です。

私たちが何気なく使うプログラムに対してCPUの中では驚異的な作業が行なわれています。

オペコードフェッチサイクル

7E
(MOV␣A,M)

CPU

01111110

この回路でCPUがどんなことをしているのか知るためには"時間"の感覚が必要です。

クロック
φ1

このICはゲート6コ入りです。

アドレス・バス

データ・バス

コントロール・バス

WAIT SWITCH

D2 8080A

D1 8224

E2 -368　E3 -368　E4 -368

D5 8216　D6 8216

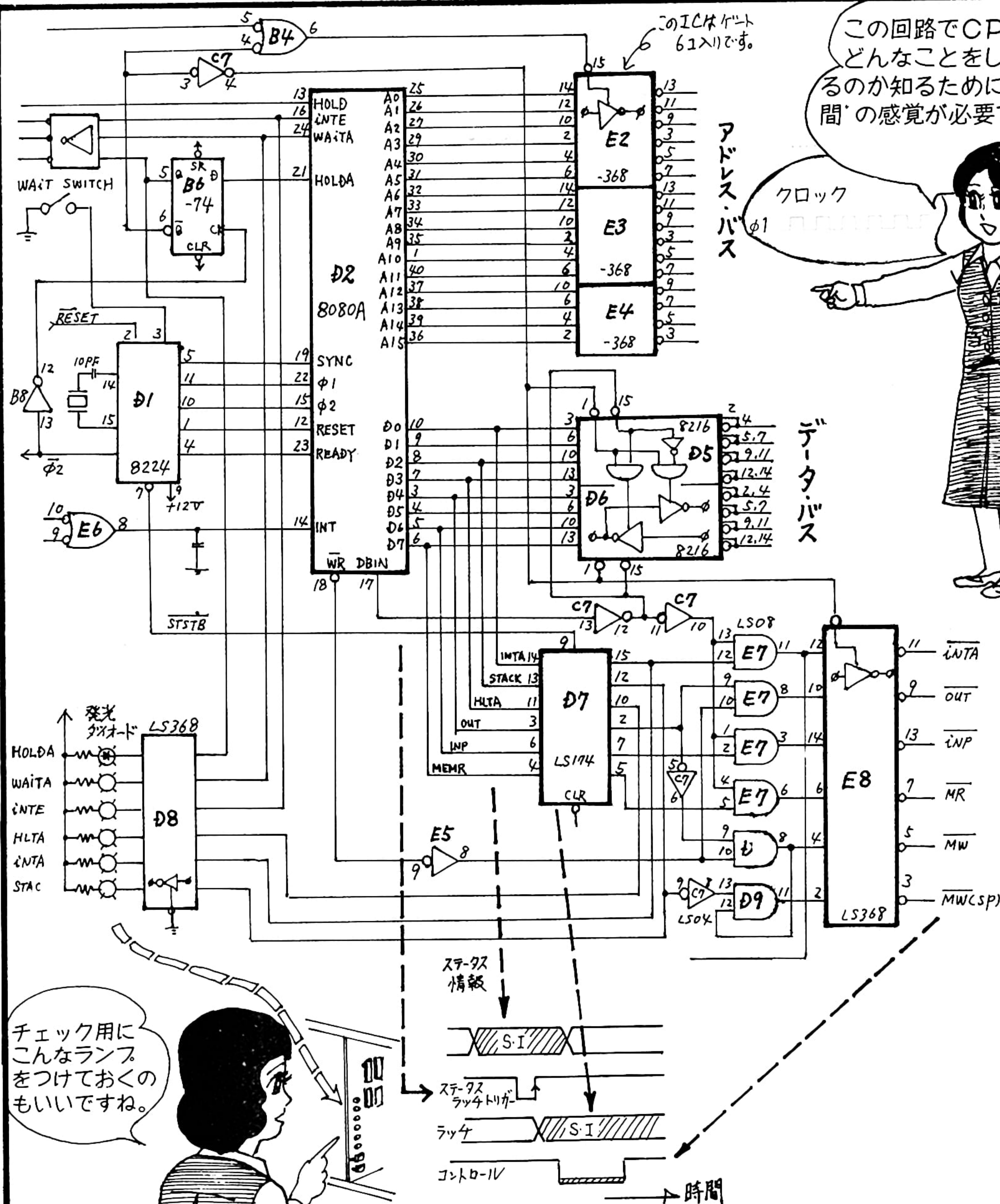
D7 LS174

E8 LS368

D8 LS368

発光ダイオード

HOLDA
WAITA
INTE
HLTA
INTA
STAC

$\overline{INTA}$
$\overline{OUT}$
$\overline{INP}$
$\overline{MR}$
$\overline{MW}$
$\overline{MW(SP)}$

チェック用にこんなランプをつけておくのもいいですね。

ステータス情報

S·I

ステータスラッチトリガー

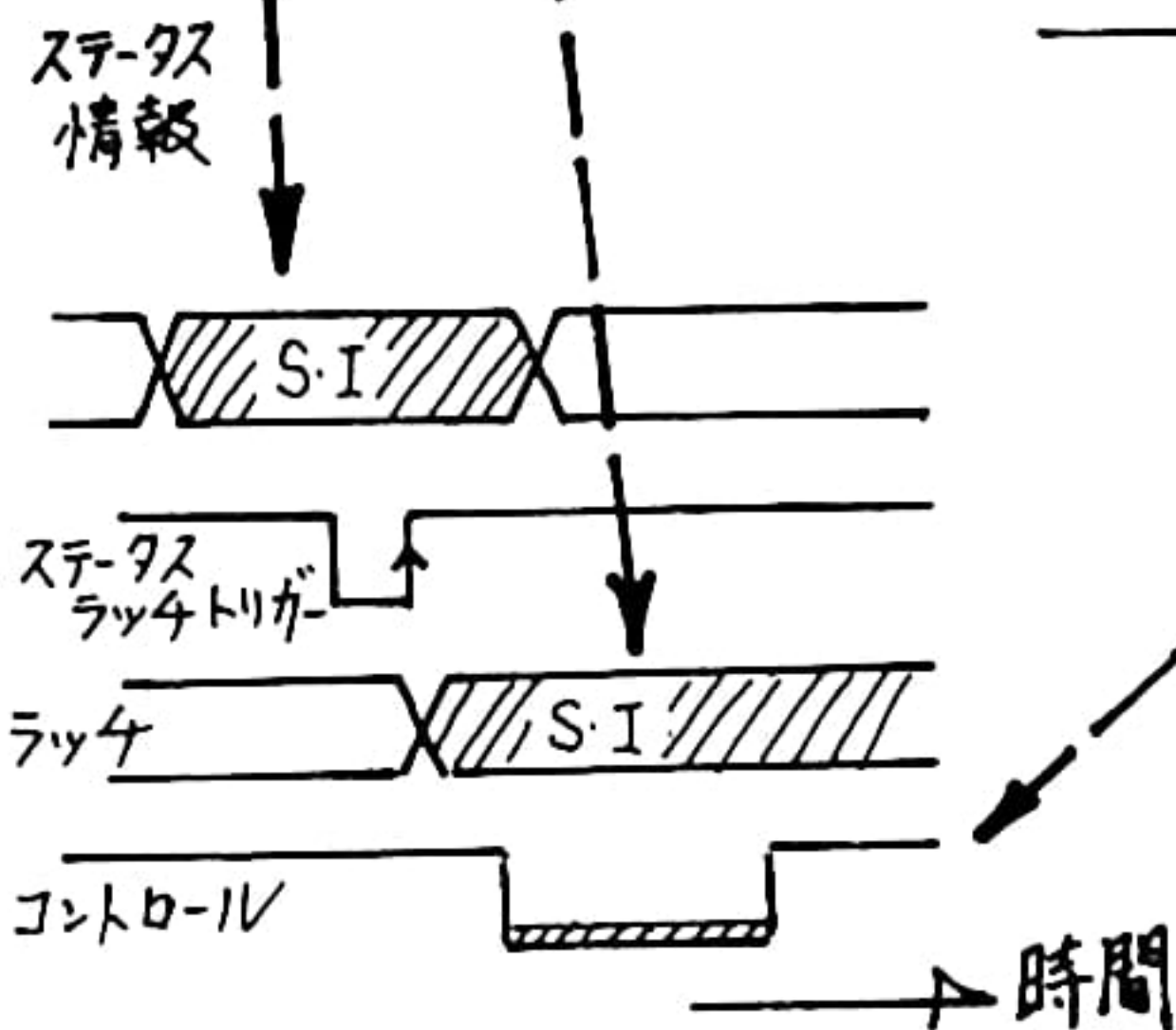
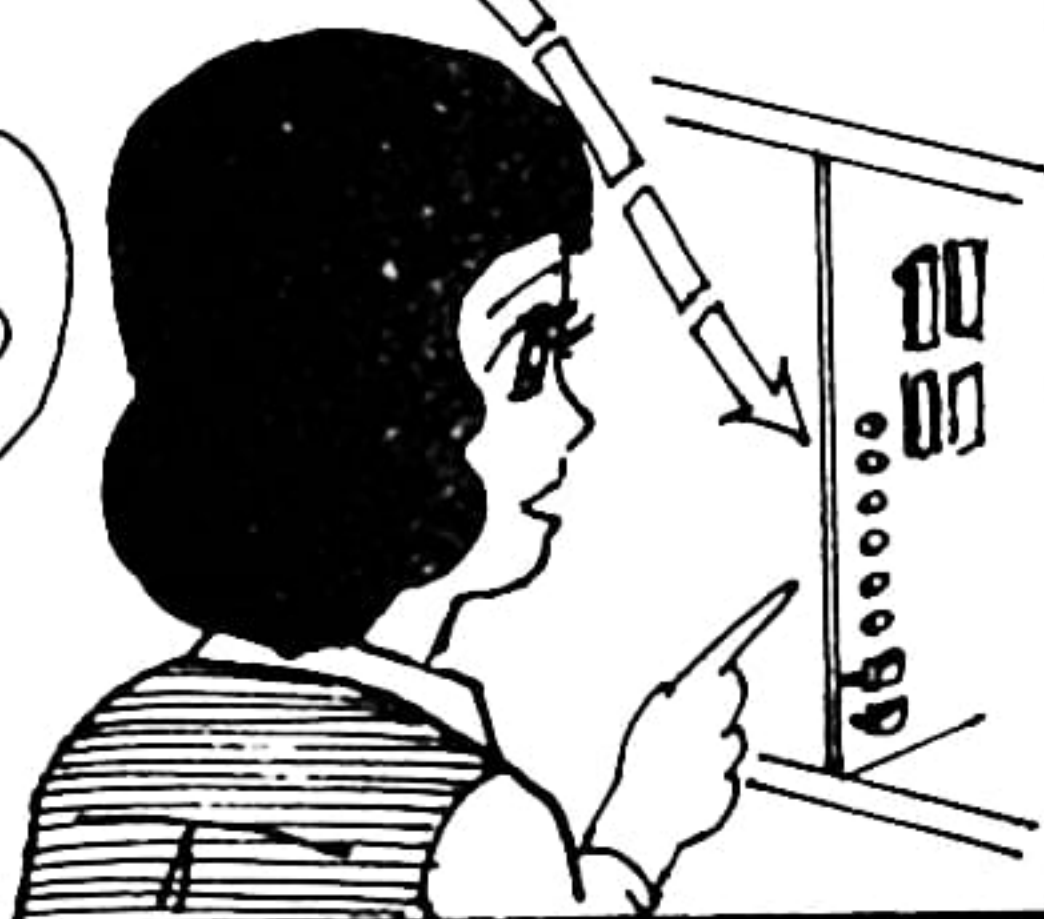
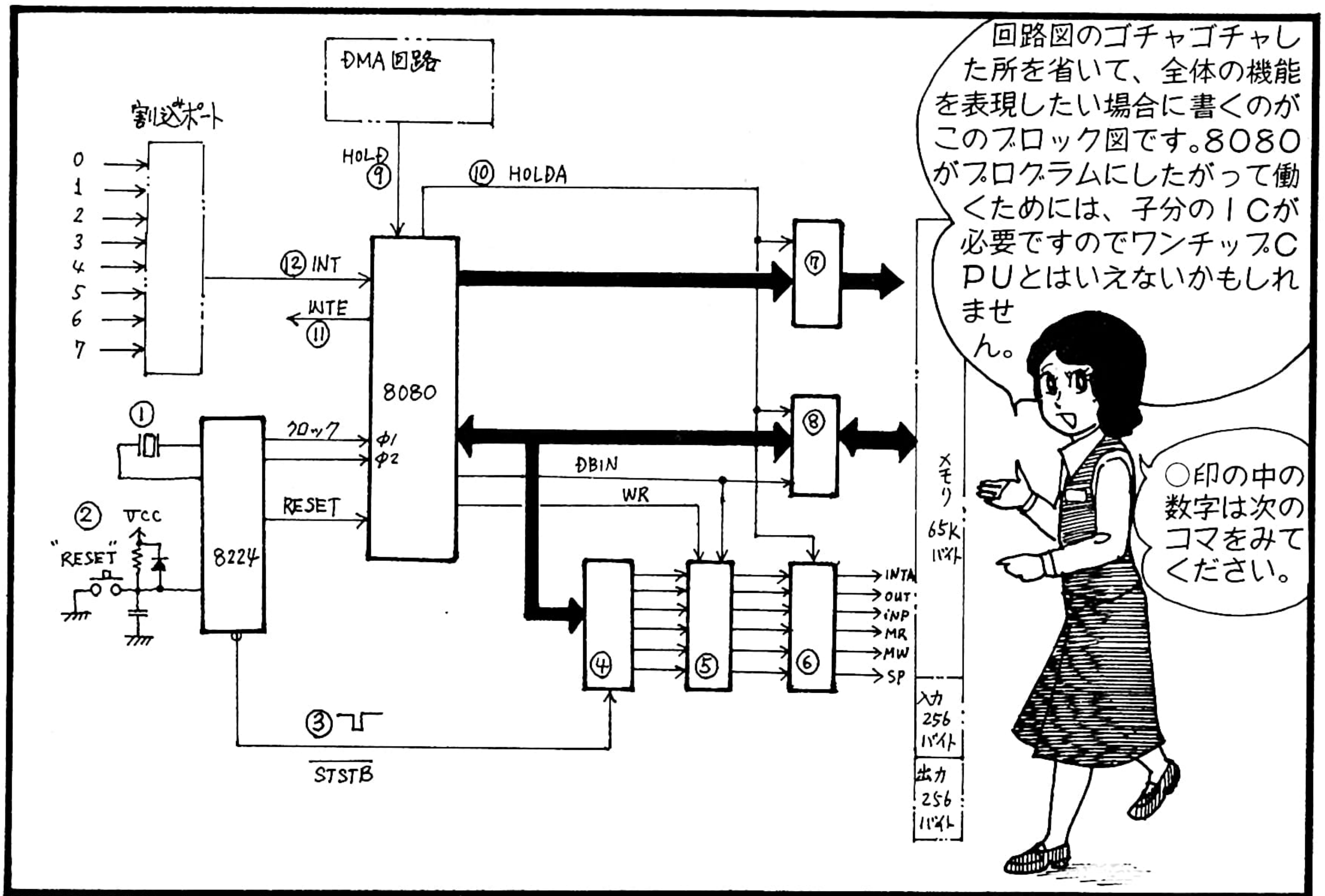
ラッチ

コントロール

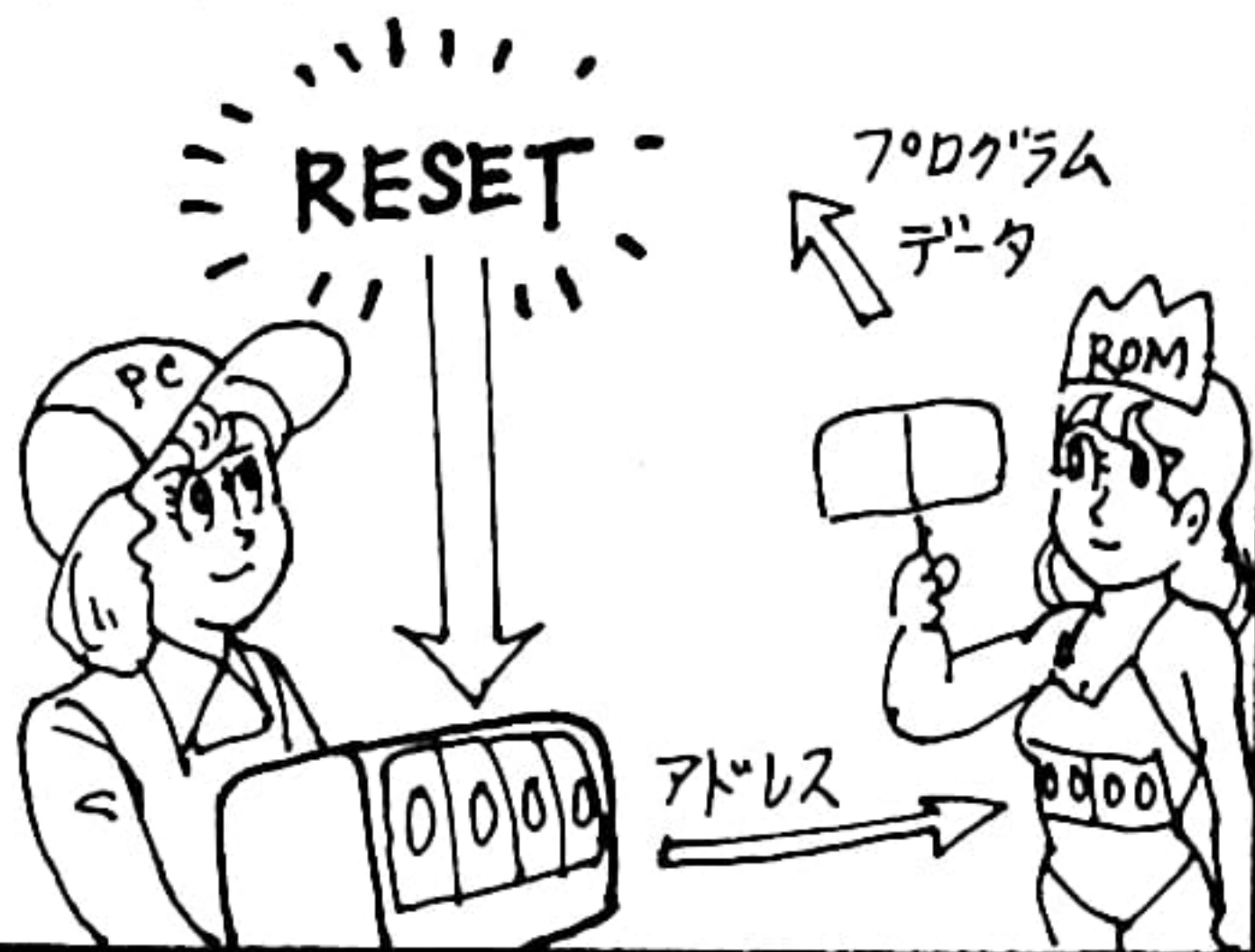
時間

DMA回路
割込ポート
0 1 2 3 4 5 6 7
HOLD ⑨
⑩ HOLDA
⑫ INT
INTE ⑪
8080
① クロック φ1 φ2
② Vcc "RESET"
8224
RESET
DBIN
WR
⑦ ⑧
④ ⑤ ⑥
INTA OUT INP MR MW SP
③ STSTB
メモリ 65Kバイト
入力 256バイト
出力 256バイト

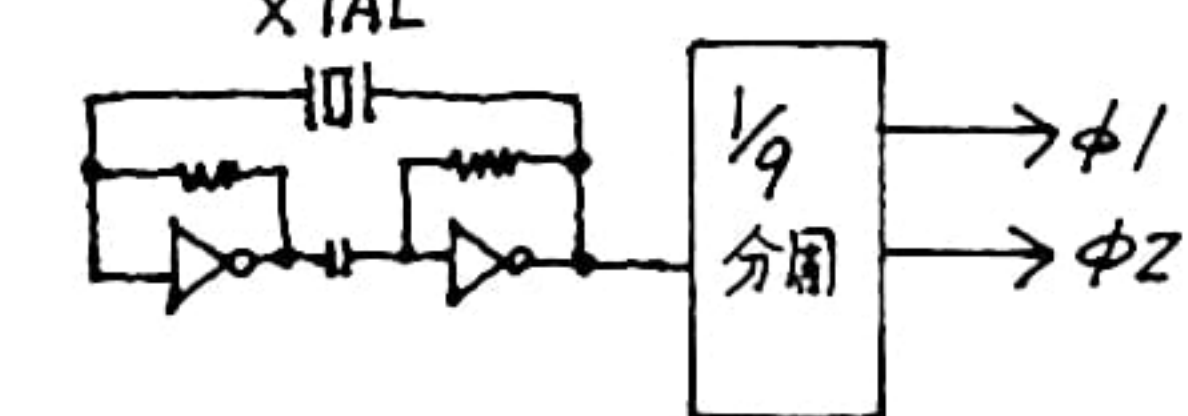
回路図のゴチャゴチャした所を省いて、全体の機能を表現したい場合に書くのがこのブロック図です。8080がプログラムにしたがって働くためには、子分のICが必要ですのでワンチップCPUとはいえないかもしれません。

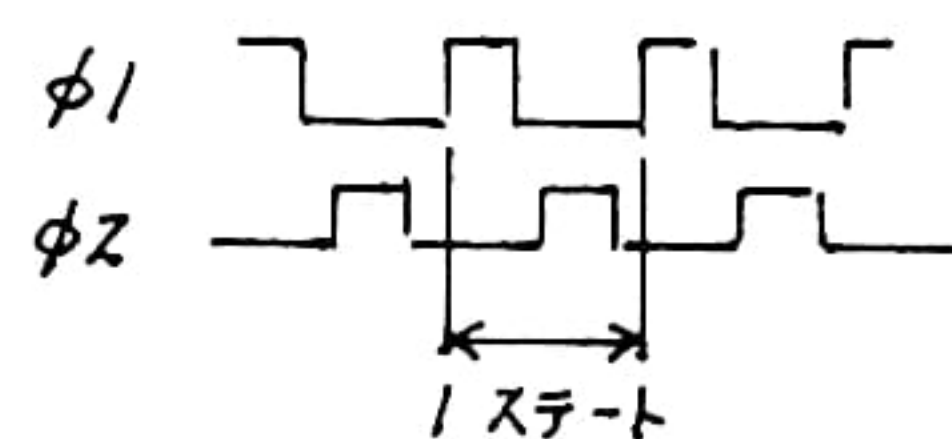
○印の中の数字は次のコマをみてください。

②RESET

この信号は、最も強力な割込みです。RESETが入るとプログラムカウンタは0000にクリアされ、プログラムはゼロ番地のオペレーションコードからスタートすることになります。

RESET
プログラム データ
PC
ROM
アドレス

①クロック・ジェネレータ

これは水晶発振器で8224と組み合わせてクロックをつくります。

XTAL
1/9 分周
φ1 φ2

このクロックの1周期を1ステートといい、マイコン動作時間の最小単位となります。

φ1
φ2
1ステート

① クロック・ジェネレータ
② 電源投入リセット
③ ステータス・ストローブ
④ ステータス・ラッチ
⑤ コントロールタイミング
⑥ コントロール(マシンサイクル)
⑦ アドレス・バッファゲート(HiZ付)
⑧ データ・バッファゲート(Hiz付)
⑨ HOLD(HiZ)要求
⑩ HOLDA受付
⑪ INTE(割込み受付可)
⑫ INT(割込み要求)

これだけわかればちょっとしたものです。

⑤コントロールタイミング

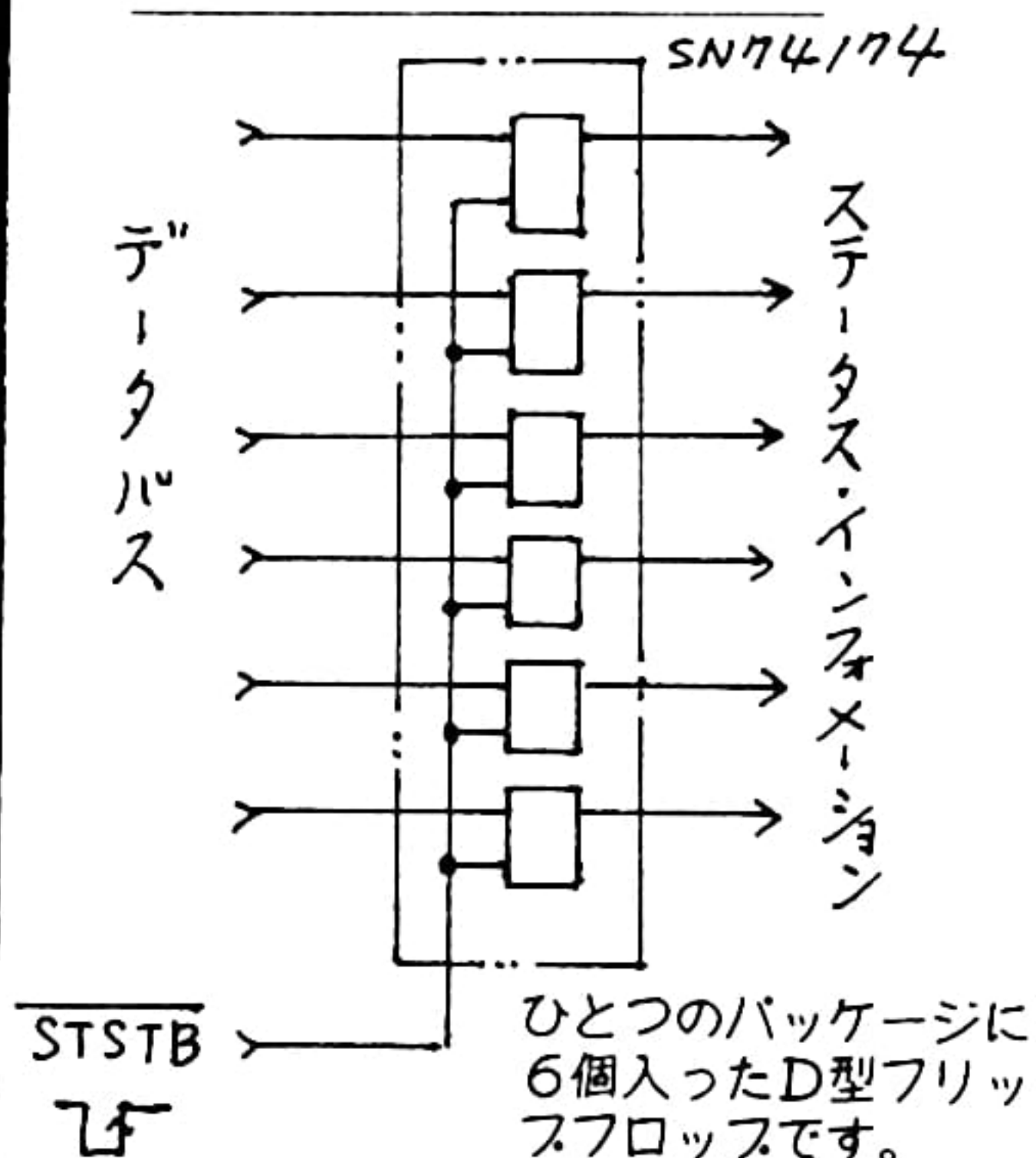
ステータス・ラッチはのっぺりしたものなのでROM、RAM、I/Oのコントロールには使えません。CPUから出力されるDBIN、WRと論理回路を組んでコントロール信号をつくります。

DBIN …… データバスがCPU読込み方向であることを知らせます。

WR …… データバスがメモリにデータを書き込むことを知らせる。

④ステータス・ラッチ

SN74174
データバス
ステータス・インフォメーション
STSTB

ひとつのパッケージに6個入ったD型フリップフロップです。

③ステータス・ストローブ

データバスにはマシンサイクルの始めにそのサイクルでは何をするのかを知らせるステータス信号というものが出ています。このステータスを外部回路でラッチさせるタイミングをとるのがこのSTSTBです。

データバス ステータス
STSTB
ステータスラッチ ステータス

⑥コントロール

この回路では６本のコントロール線がありますが、各々必要なデバイスに接続されます。

INTA
OUT
INP
MR
MW
SP

メモリ・デバイスにはMR、MWでコントロールします。

アドレス　ROM　データ
(HiZを解除)
MR
アドレス　RAM　データ
MW
SP
データの方向
データバス

INPUT、OUTPUTの場合はinp、outを使います。

アドレス　インプットポート　データ
INP
アドレス　アウトプットポート　データ
OUT
データバス

INTAはインタラプト・アクノリッジの略で割込み要求が受け付けられたことを示し、このマシンサイクルでデータバスの内容をRST 命令として読み込まれます。

RST命令挿入
11NNN111
0
1
2
3
4
5
6
7
割込みポート
INTA
データ・バス

⑧HOLD ホールド要求

この信号は、アドレス・データ・コントロールの３つのバスをＣＰＵの占有から切り離してほしいという外部ハードからの要求です。

HOLD
8080

⑩HOLDA

ホールド・アクノリッジホールド要求を受容した確認信号で、CPUはプログラムの実行を停止するとともにこの信号を出し続けます。回路図ではこれを３つのバスのHiZゲートに入れています。

HOLD
HOLDA
8080
HiZ
HiZ
HiZ
システムバス

⑪INTE

インタラプト・イネーブル

この信号は、プログラムでＥＩ命令を実行させる必要があります。これが出ている時はCPUが割込み受け付け状態にあるということです。

INTE
8080

⑫INT

インタラプト要求

外部から回路の割込み要求端子です。CPUがINTE状態にあると要求を受け付け、次のマシンサイクルでステータスにINTAを出力します。

INT
8080

INPUT、OUTPUT、INTについては、メモリ65Kバイトのほかに、ポートという256、256、1バイトの専用メモリエリアがあるのです。

メモリ 65K	インプット 256	アウトプット 256	割込み 1

この３つのポートについては、8080タイミング説明とは別にお話することにします。

インプット ポート
アウトプット ポート
インタラプト ポート

では、8080の内部に出発です。最も必要なものは時間に対する感覚ですヨ。

タイミングチャート
8080
CLOCK

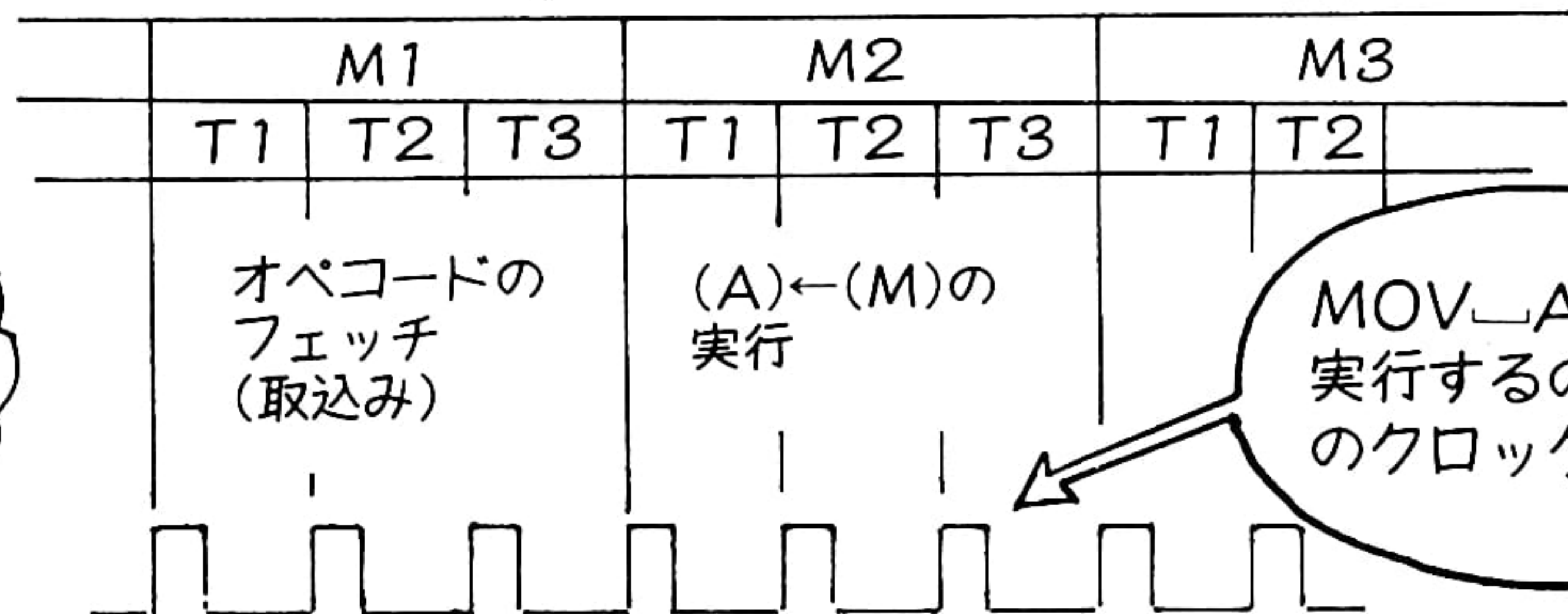
⑨CPUとクロック

あら？！
大変

MOV␣A，Mの実行

M1…オペコードフェッチサイクル
M2…実行サイクル

M1			M2			M3	
T1	T2	T3	T1	T2	T3	T1	T2
オペコードのフェッチ（取込み）			(A)←(M)の実行				

MOV␣A，Mを実行するのに6つのクロックが必要

この図形化されたクロックの波形を見る時、次のことにまどわされないようにしましょう。

クロック

AC100V

シンクロスコープで見ると、回路に　　　のような電圧の変化があることがわかります。

回路的には、電圧が規則正しく現われたり消えたりします。

クロック

5V---

0V---

タイミングチャートからは、ある時間のできごとを読み取るんだと思うと楽ですよ。

CLK

A

B

C

シンクロスコープの場合も、ひとつの☼が動いているだけですが、人間が見ると連続したものになります。

それは、ある時間には一点にしか☼がないということです。

5V

0V

この端の急激な変化は、微分回路でトリガーをつくることができるからです。　この図はクロックとトリガーを併記したものです。

この端が、回路的には重要です。

クロックを拡大すると、ひとつの山があり、両端があります。

外からきた信号をクロックと同期させ、マイコンが扱いやすい信号に整形することができます。

クロック
外部信号
同期信号

クロックのひとつの繰り返し時間を周期Tで表わし、

T

1秒間に何周期あるかをHz(ヘルツ)という単位で示します。図の場合は、6Hzとなります。

1 2 3 4 5 6
T
1秒

マイコンのクロックは1MHzから4MHzです。

RUN
よーい どん あれ？
ズドドドドッ
マニュアル

1MHz(メガヘルツ)は、100万Hzのことです。パルスの幅は$\frac{1}{100万}$秒です。

$\frac{1}{100万}$秒

マイコンでは音を出すことができますが、秘密はこのクロックの速さにあります。

コンピュータの演算速度はクロックが何Hzかによって決まってきます。

やるぞー

$Hz=\frac{1}{T}$

このクロックは勝手に決めて良いわけではなく、CPU、RAM、ROMの処理スピードを考えて決められます。

ROM
仲よく走ろう
ROMちゃんには負けるわ
RAM
8080

メモリから読み込む時、また書き込む時、内容が違って伝達されたらなんにもなりません。

ROM
RAM

8080 CPUは、外部にクロック回路をつける必要がありますが、最近のものは、CPUチップに内蔵されているものもあります。

クロック回路 → 8080 CPU

マイコンチップの内部では、クロックが入るたびに、何らかの動作をしているのです。プログラムがなくても動作しています。

H
L
NULL
NULL
NULL
NULL
⋮

コンピュータはクロックで動作します。

クロックの周期は、水晶発信器の周波数によって決まります。
8224によって$\frac{1}{9}$に分周されます。
TK-80
18.432 MHz
回路図の記号

周波数が高い、つまり周期が短いほどCPUの処理スピードは速くなるわけですが、
N:周波数
T:周期
$N=\frac{1}{T}$

外部メモリの、応答速度によって、主としてこの周波数が制限されています。
CPU自身の限度もあります。
うーんと
OUT PUT
IN PUT

この場合はOUT1の方がOUT2よりも、応答速度が速いということができます。
IN
t1
OUT1
t2
OUT2
時間

自分がこれから使おうとしているメモリや周辺機器の応答速度を念頭に入れて、より速くて確実な処理系を作っていくのがハード設計者の腕の見せどころというものです。

8080はφ1、φ2の2相のクロックで動作します。
ファイ
φ1
φ2
H
L

φ1がHのサインを出している時は、
H
φ1

φ2はLのサインを出しています。
φ2
L

この波形からφ1、φ2が同時にHになる時間は存在しないことを読みとってください。
φ1
φ2
H
L
時間

これは、データを不安定な所で読んだり、書いたりしないためなんです。

1相のクロックの場合
ここでデータチェンジの指令
ここでデータを読み書きする
コントロール
データチェンジ
データ(旧)
(新)
データ不安定
時間

メモリの読み出し時間、また書込み時間に固有の長さやバラつきがあります。

MOV␣A，Mのオペコードは、01111110(B)つまり7E(H)です。(なんのことかわからない人は前に戻ってください)。

このOPコードのフェッチから実行までのハードの動きを追ってみます。

回路図やプログラムの中に時間関係が見えてきたらコンピュータの知識は、かなりのもの、ということでしょうか。

φ2

φ1

コンピュータは自分がうまく動けるように時計をもっています。私たちもそのことをよく理解していないとかえってわからなくなるのです。

コンピュータの時計は、クロックと言っています。

私は16ビットのラッチです。

私、アドレス・レジスタはPCの内容をアドレスバスにのせていますのでCPUはこの2073番地のメモリの内容をOPコードとしてフェッチします。

20 73

アドレス

PC

プログラムデータ読出し独占

私、プログラムカウンタは今、2073になりました。

PC

20 73

メモリ

アドレス(番地)

2073 7E

プログラムエリア

21FF 60

データ・エリア

このOPコードは、プログラム・エリアの2073番地に入っています。

私はホールドアクノリッジ、HDLAパルス担当です。

私は、ライトちゃん、WRパルス担当です。

私たちはCPU 8080のまわりの治安を担当しています。

私は、データイン、DBINパルス担当です。

私はシンクロナス、SYNCパルス担当です。

さて、ここで制御担当の新しい仲間をご紹介しましょう。

HDLA

WR

DBIN

SYNC

このSYNCパルスの長さはφ2クロック1周期分にあたります。

φ2

SYNC

実際にはφ2よりも少し遅れます。

→時間の経過

私は、マシンサイクルの最初を知らせるためSYNCというパルスを出します。

SYNC

上げて降ろしてまた上げるまでの時間SYNCを点灯するの。

φ2

シンクロナスとは、同期という意味です。

私は、直接コントロール信号には関係していません。

SYNC

SYNCちゃんの言ったマシン・サイクルについて補足しましょう。

これは、MOV␣A,Mの内容を説明するために書いた719コマの図です。実は、これがマシンサイクルM2に当たります。

私たちは、実行過程でのアドレス指定に使われます。

データエリア

RAM

80

データ転送

A

H L

21 FF

アドレスバッファ

アドレス指定

ソフトの立場から命令の意味を理解するのならこのひとコマで十分なんですが、

MOV A,M

ハードの立場から見ると、

もっと細かく見ていく必要があります。

OPコード読込み　実行

すべての命令について共通しているのは、CPUは、まず外部のプログラム・メモリから命令コードつまりOPコードを読んでくることが必要です。

時間の経過

01・
00
20
21・
03
20
AF・

・がOPコード

この命令を読み込むサイクルをマシンサイクルM1といいます。

M1は OPコードフェッチ サイクル

マシンサイクルM1を表わすと、だいたいこんな様子です。

マシン・サイクルM2へ移る

OPコード・レジスタ

7E

データバス

プログラム・エリア

ROM

20 73

アドレスバス

たとえば命令の7Eをフェッチすると、HLの内容をアドレス・バッファに入れて、メモリのアドレスを指定しMの内容をAレジスタに入れます。この段階をマシンサイクルM2といいます。MOV␣A,MはM1、M2のマシンサイクルで実行を完了します。

マシンサイクルの数は、それぞれの命令によって決まっています。

私はφ2の一周期の間だけスイッチを入れますが、この長さでは使われません。

φ1
φ2
SYNC

クロック・ジェネレータ8224でステータスストローブSTSTBとして使われます。

8080 CPU

8224

SYNC

STSTB

ステータスは、SYNCパルスと同じ時間にデータバスに現われます。

φ2
SYNC
DATA BUS 8Bit
ステータス
8ビットのデータのチェンジを表わします。

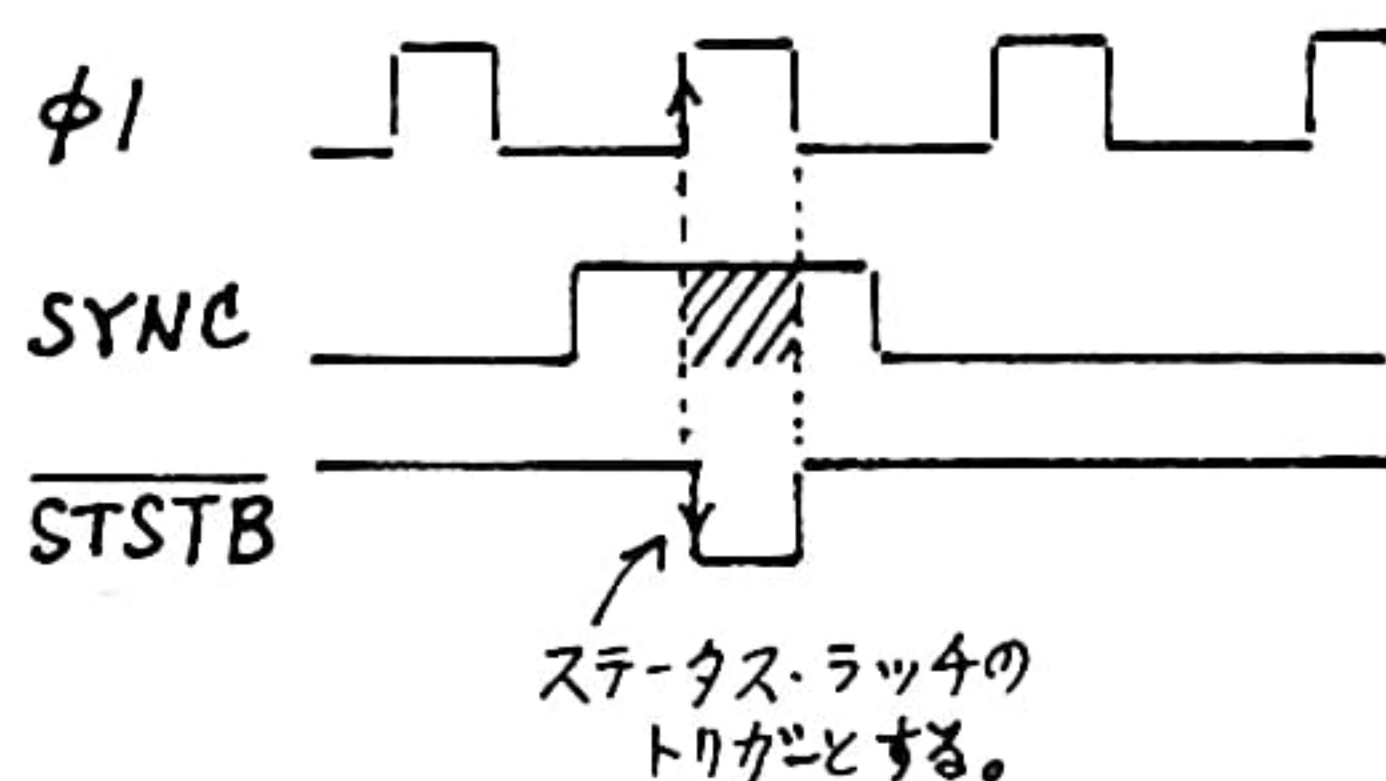
このステータス情報は、データバスからは、すぐに消えてしまうのでタイミングをとって外部回路でメモリする必要があります（データではないのでメモリとは言わず、ラッチすると表現します）。

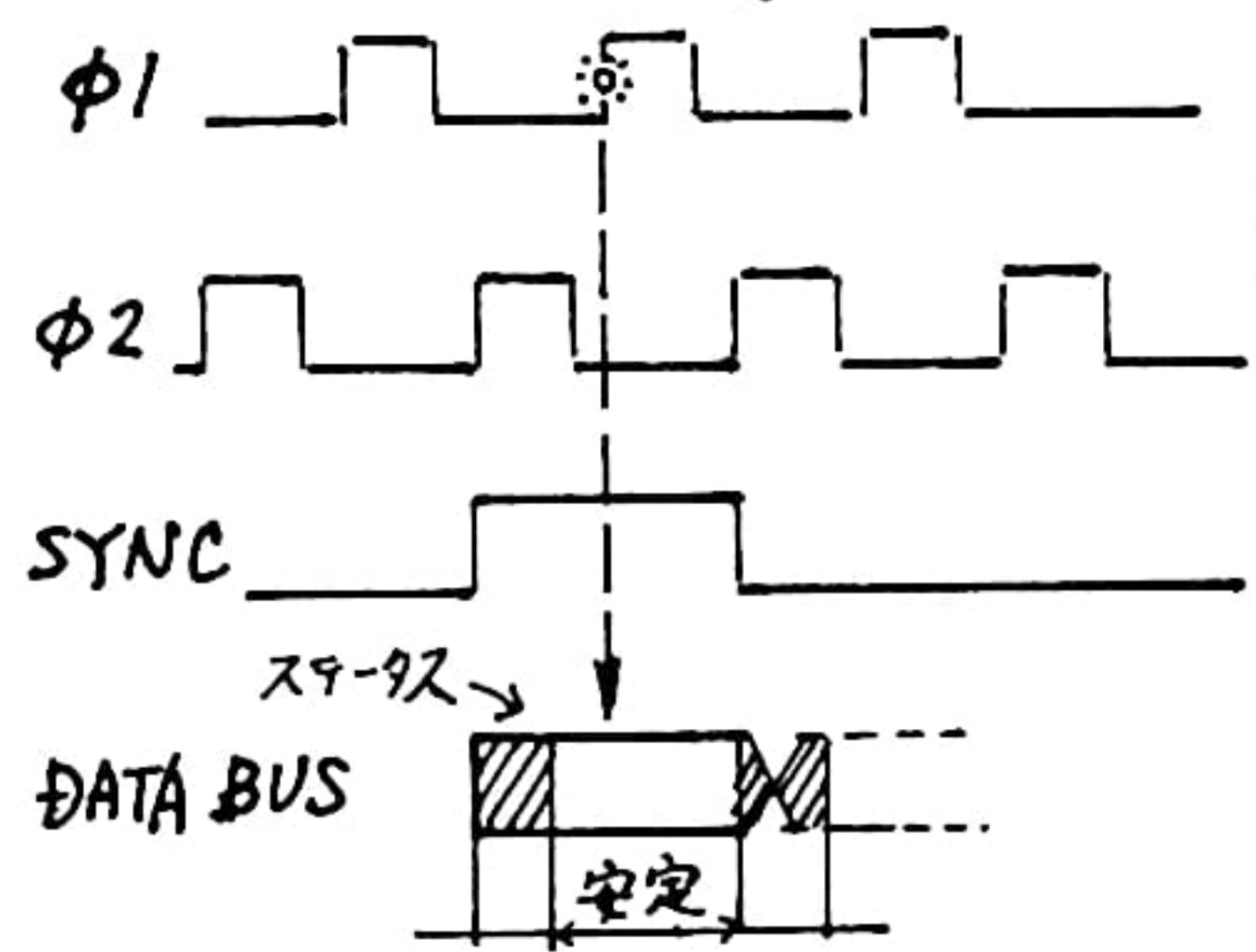
SYNC
ラッチ 電子的に鍵をかけて信号を保持すること。

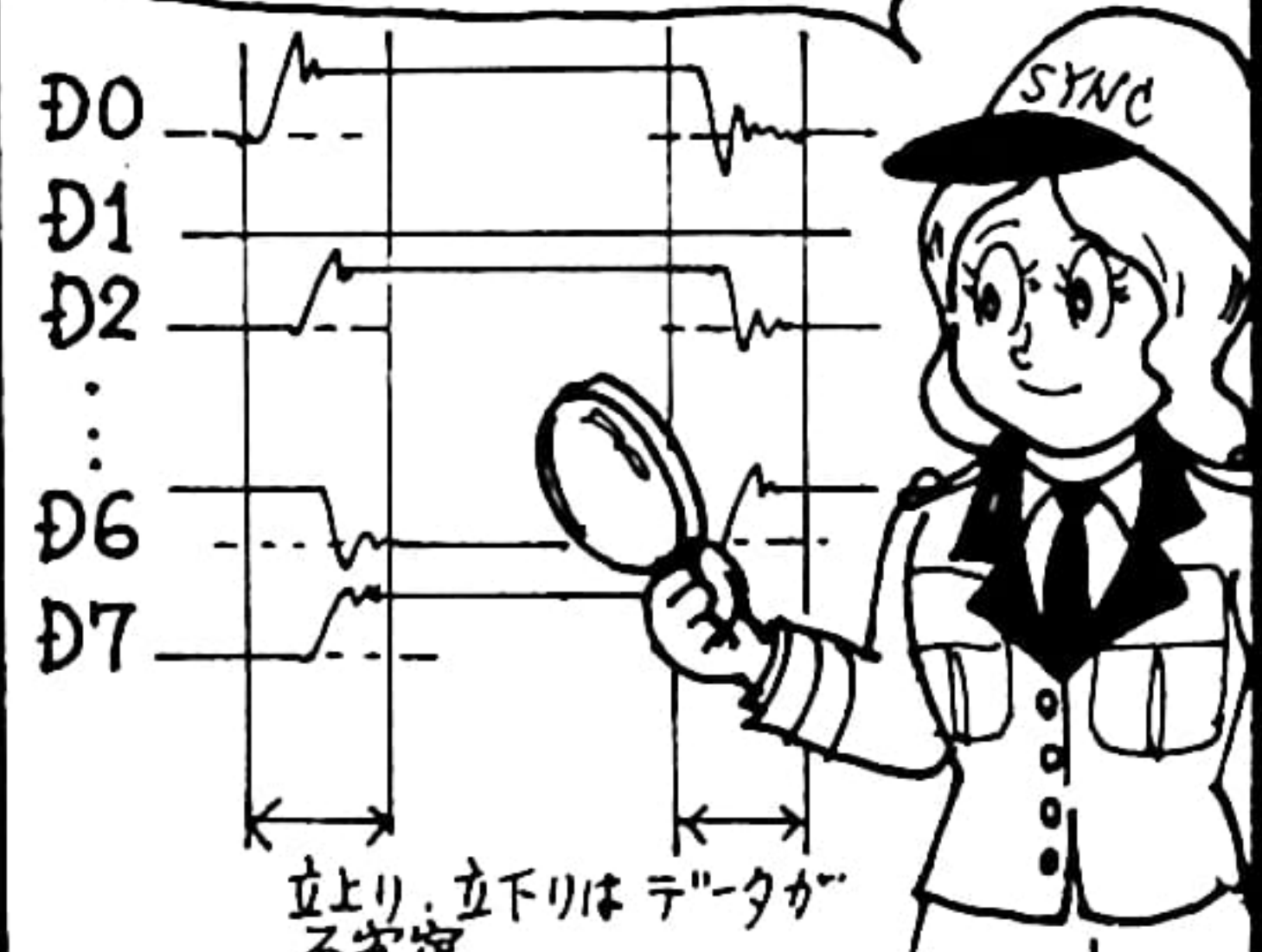
マシンサイクルの最初にCPUからはデータバス8ビットにステータス情報が出力されます。

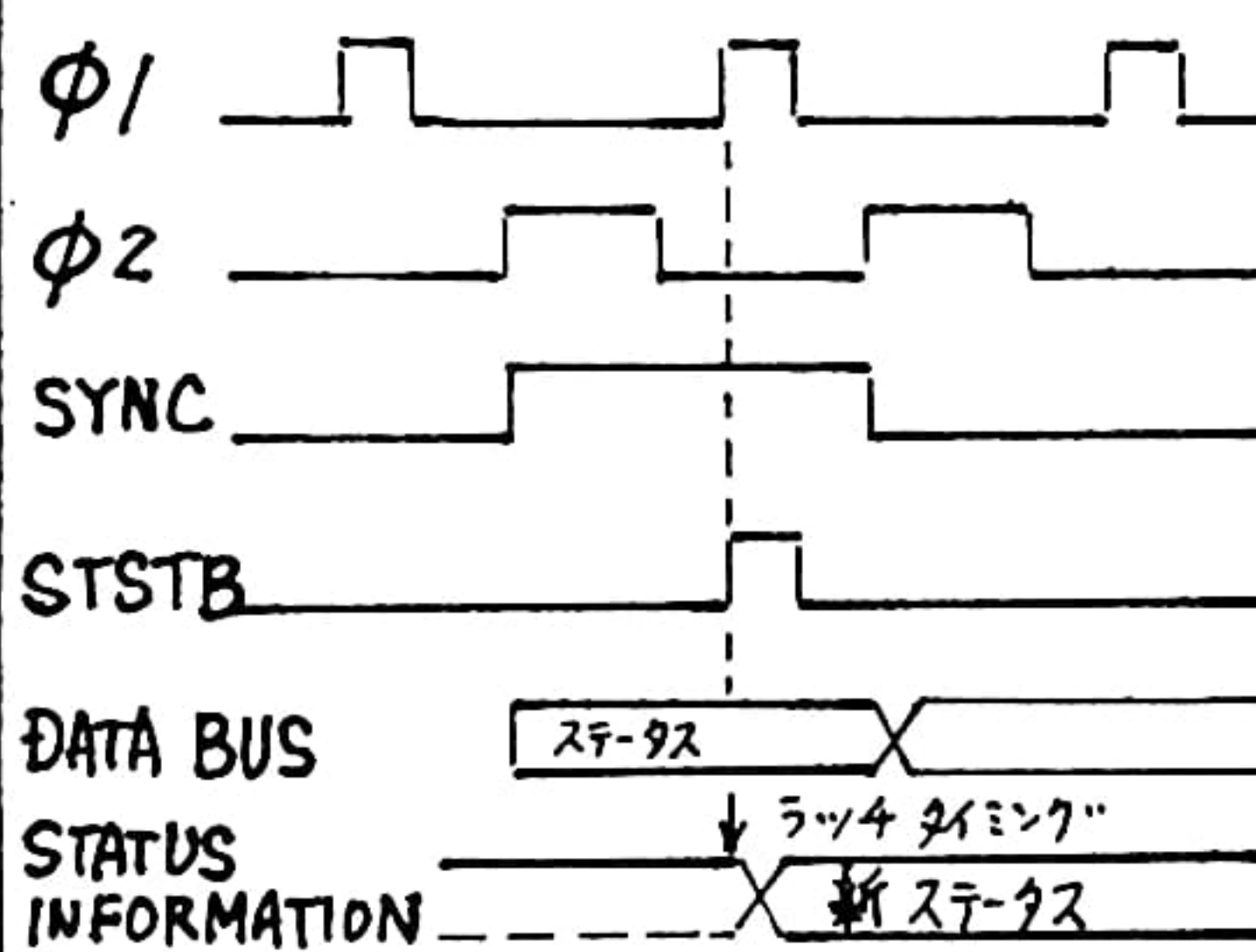
CPU
SYNC
ステータス D0〜D7
SYNC
I/O
RAM

ここで、φ1に注目すると、φ1のエッジはデータの安定域にありますね。φ1は次々とくるので、SYNCとANDをとって$\overline{\text{STSTB}}$としして、これをトリガーとします。

φ1
SYNC
$\overline{\text{STSTB}}$
ステータス・ラッチのトリガーとする。

正確にデータをラッチするためにはパルスの両端の不安定な時間帯でのトリガーを避けなければなりません。

φ1
φ2
SYNC
ステータス
DATA BUS
安定

データバスは8ビットありますから、各ビットごとの応答時間にはバラツキがありエッジの部分の波形はみだれています。

D0
D1
D2
D6
D7
SYNC
立上り、立下りはデータが不安定

教科書にはこれらのことが、ひとつのタイミングチャートに表わされています。

φ1
φ2
SYNC
STSTB
DATA BUS
ステータス
STATUS INFORMATION
ラッチ タイミング
新ステータス

$\overline{\text{STSTB}}$は、$\overline{\text{STATUS STROBE}}$の略です。これと、SYNC、φ1との関係はNANDゲートのインプットとアウトプットになっています。

SYNC
φ1
$\overline{\text{STSTB}}$

この$\overline{\text{STSTB}}$とタイミングをとって、外部メモリにデータバスの内容をラッチすると、それがステータス情報として、次に送られるデータの処理に使われるわけです。

$\overline{\text{STSTB}}$
DATA BUS
ステータス
STATUS MEMORY
外部メモリ
旧
新しい ステータス

これからはS.Iと略します。このS.IはCPUが次のマシンサイクルで何をするつもりなのか外部に知らせるためのものなのです。

Nop
8080
書きたい
読みたい
入力したい
出力したい
割込み受け付けたい
その他5つ

8080ではSYNCのタイミングでデータバスにステータス・インフォメーションというものがのっている、ということです。

8080
データバス（ステータス・インフォメーション）
SYNC
STSTB → ラッチ

私は、このSYNCパルスをマシンサイクルごとに送っています。では友子ちゃんにバトンタッチしましょう。

8080
8224
SYNC
データ・バス
$\overline{\text{STSTB}}$
メモリ
ステータス

8080では、このデータバスにマシンサイクルの最初にS・Iをのせます。D0～D7にそれぞれ意味を持たせます。

ステータスのラッチ

8080

STSTB

データバスはD0～D7までの記号で表わされます。D0が最小ビット、D7が最大ビットを示します。

CPUのデータバスは8ビットです。

8080端子図

S.I は SYNC に同期してデータバスに現われる！

MEMR INP M1 OUT HLTA STACK $\overline{WO}$ INTA

D7 D6 D5 D4 D3 D2 D1 D0

ステータス・インフォメーション（S・I）はデータではありません。それぞれのビットごとに意味があるんです。

私D2は、スタックとのデータ転送を知らせるのが仕事です。アドレス・バスにスタックポインタの内容が入ってるよ、って言ってやるのよ。

SP

STACK

私D2はCPUがこのマシンサイクルで外部にデータを書き込むことを知らせるためのものです。私は'0'。'1'の時は読み込みです。

"0"の意味

$\overline{WO}$

私D1は「割り込み許可」を外部に知らせるのが仕事です。INTAはインタラプト・アクノリッジの略です。

INT-A

私D5はCPUがOPコードの1バイト目を読み出すマシンサイクルであることを知らせます（M1サイクルのM1です）。

M1

私D4は、I/O命令のうちのOUTPUTが実行されることを示します。

OUT

私D3は、CPUがHALT命令を実行して停止状態に入ったことを知らせるの。

よきにはからえ、ワシャひと休み。

割込みがくるまではおこすなよ。

HLTA

私D6は、I/O命令のうちINPUTが実行されることを示します。

INP

私D7は、データバスがメモリからの読み出しのために使われることを知らせます。

MEMR

これら8ビットを組み合わせたマシンサイクルの形態は10種類です。これ以外はありません。

マシンサイクルは10種類

続けてそのマシンサイクルの形を紹介します。

TYPE 1（OPコード読込み）
INSTRUCTION FETCH

D7 D6 D5 D4 D3 D2 D1 D0
1 0 1 0 0 0 1 0

OPコードを要求する

CPU　データ　RAM　ROM

TYPE 2（メモリ読込み）
MEMORY READ

D7 D6 D5 D4 D3 D2 D1 D0
1 0 0 0 0 0 1 0

メモリを読込むよ。

CPU　データ　RAM　ROM

TYPE 3（メモリ書込み）
MEMORY WRITE

D7 D6 D5 D4 D3 D2 D1 D0
0 0 0 0 0 0 0 0

読出専用

データをメモりたい

CPU　データ　RAM　ROM

TYPE 4（スタック読込み）
STACK READ

D7 D6 D5 D4 D3 D2 D1 D0
1 0 0 0 0 1 1 0

スタックからのメモリ読込み

CPU　データ　RAM

TYPE 5（スタック書込み）
STACK WRITE

D7 D6 D5 D4 D3 D2 D1 D0
0 0 0 0 0 1 0 0

スタックへのデータ書込み

CPU　データ　RAM

TYPE 6（インプット読込み）
INPUT READ

D7 D6 D5 D4 D3 D2 D1 D0
0 1 0 0 0 0 1 0

I/Oからのデータ読込み

CPU　データ　I/O

TYPE 7（アウトプットポート書込み）
OUTPUT WRITE

D7 D6 D5 D4 D3 D2 D1 D0
0 0 0 1 0 0 0 0

I/Oへのデータ書込み

CPU　データ　I/O

TYPE 8（割込み応答）
INTERRUPT ACKNOWLEDGE

D7 D6 D5 D4 D3 D2 D1 D0
0 0 1 0 0 0 1 1

割込受付けみた。

データ　割込み　INT

TYPE 9（HALT応答）
HALT ACKNOWLEDGE

D7 D6 D5 D4 D3 D2 D1 D0
1 0 0 0 1 0 1 0

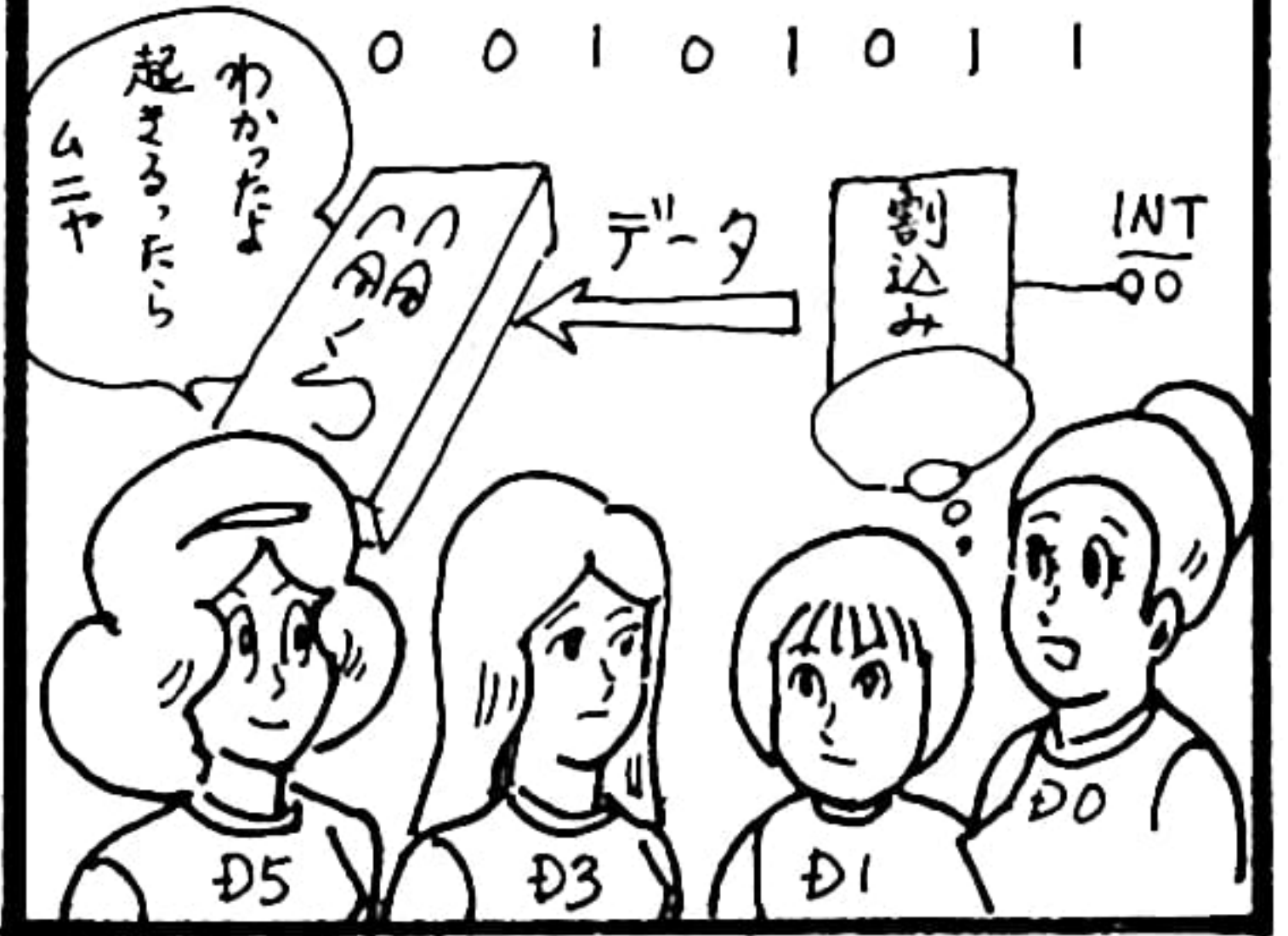
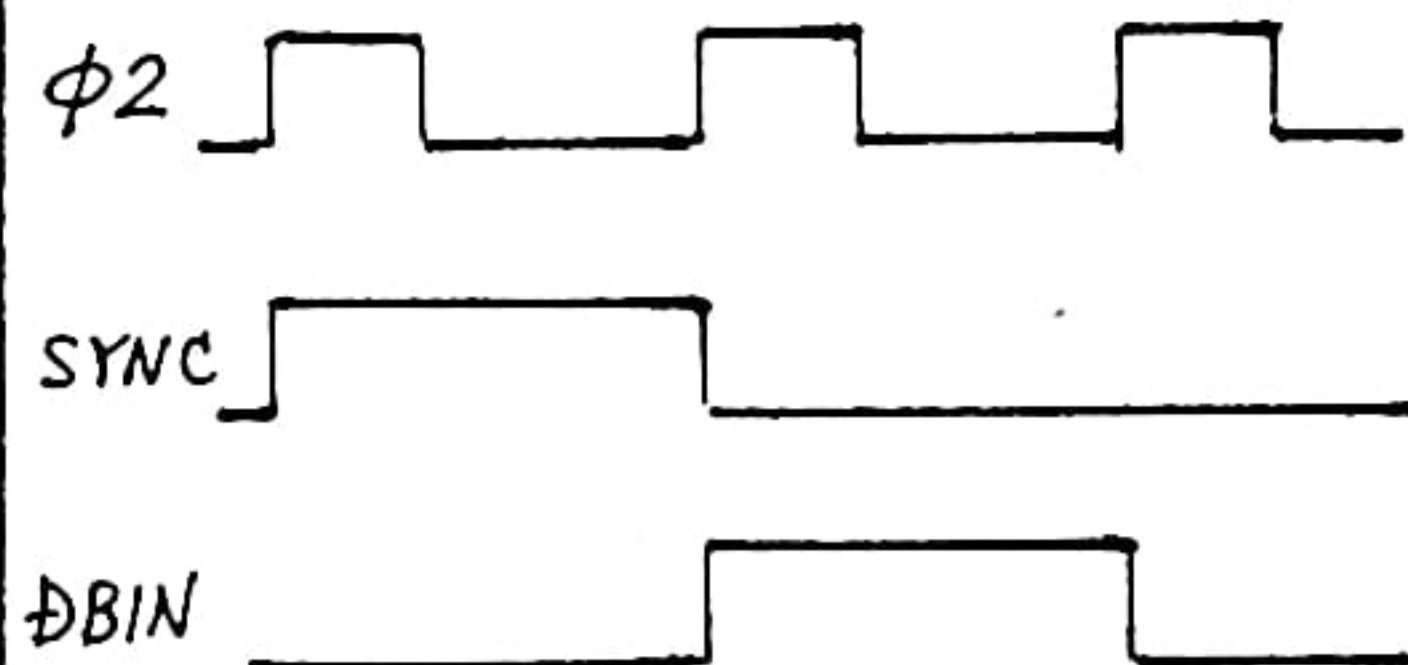
停止させてかって よっしゃ

CPU

TYPE 10（HALT中の割込み応答）
INTERRUPT ACKNOWLEDGE WHILE HALT
D7 D6 D5 D4 D3 D2 D1 D0
0 0 1 0 1 0 1 1
わかったよ 起きるったら ムニャ
データ
割込み
INT
D5 D3 D1 D0

ひとつの命令を実行するためにいくつかのマシンサイクルが繰り返されるわけですが、CPUからは常にこの10種の中のひとつのS・Iが最初に出力されているわけです。
SYNC

次に、このステータスを使ってコントロール信号をつくり、データを外部メモリに書き込んだり、内部レジスタに読み込んだりしなくてはなりません。
ステータス → コントロール

SYNCの次は、私DBINです。私は、双方向性であるデータ・バスが、どっち向きになっているのかを知らせるんです。
DBIN
只今のデータバスは 入力

タイミングとしてはSYNCちゃんのすぐあとに出ます。
SYNC
DBIN

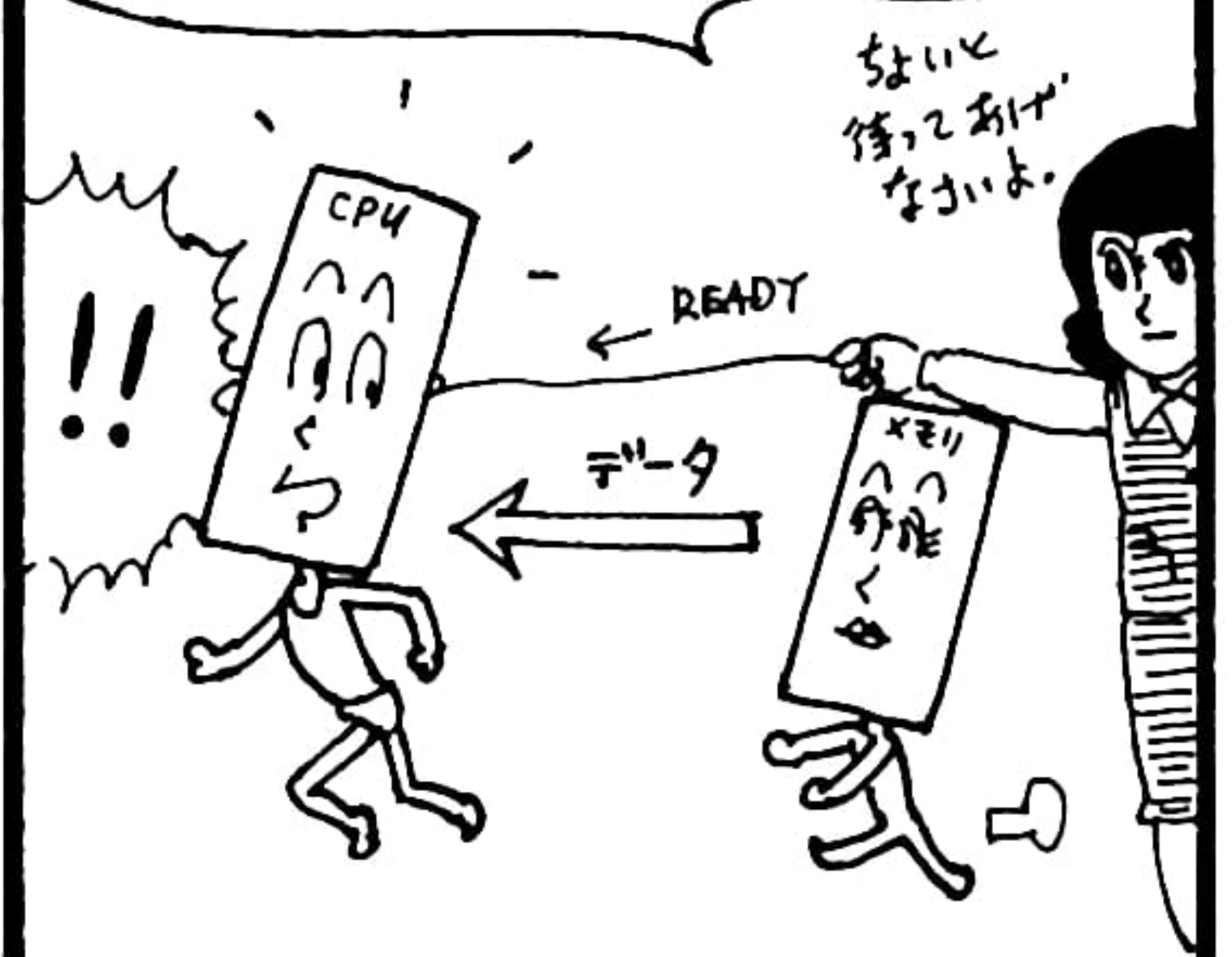
長さは普通SYNCと同じ、ϕ2のクロック1周期分ですが…
ϕ2
SYNC
DBIN

READYがくると、ϕ2の整数倍のクロック分だけ長くなり、足ぶみ状態になります。
ちょいと待ってあげなさいよ。
!!
CPU
READY
データ
メモリ

マシンサイクルは、さらにクロック単位でT1、T2、T3といったステートに分けられます。
T1 T2
ϕ1
ϕ2
DBIN

DBINは、T2～T3のステートの間出力されるということです。
T1 T2 T3 T4
ϕ1
ϕ2
SYNC
DBIN
データ読込みのタイミング
CPUが勝手に読込む

私はDBINの信号を出すだけでなく、T2のステートでREADY信号がきていないかも見ているわけです。
T1 T2 TW T3 T4
READY
DBIN

もし、READY信号がT2ステートでLOWになったら、何もしないTWというステートにし、CPUを足ぶみさせ、T3ステートに移らないようにします。
T1 T2
READY
DBIN

これはアクセスの時間の長いメモリなどの、応答待ち用として考えられた機能なのです。
足ぶみ
T1 T2 TW TW TW T3 T
ϕ1
ϕ2
SYNC
DBIN
READY
CPUがデータを読込む

メモリへのデータ書込みはWRパルス全体を使います。TTLメモリのラッチよりも時間がかかるんです。スタティックRAMで250n(ナノ)sec以上のパルスを入れる必要があります。

データバス
WR
この時間内にRAMのデータセット
書込250nsec以上

私はふつうT3ステートの間WRを出力するんです。

M1 M2
T1 T2 T3 T1
φ1
φ2
SYNC
WR
データ
アドレス
アクセス時間

私はライト・メモリへの書き込みタイミングです。DBINおねえさんよりタイミングは少し遅くなっています。DBINと同じマシンサイクルの中でいっしょに出ることはあり得ません。

8080
データバスの方向
OUTモード
WR

TK-80では、WAITとREADY端子を直結してONE WAIT動作をさせています。

WAIT
READY

READYがきていると、TWステートに入り、T3ステートに移りません。これは最初ハード設計の時クロック周期や、メモリ速度をもとに何回待つか決めるんです。

T2 Tw Tw T3
WR
READY
8080

私もT2ステートではREADYがくるのか見ています。

T1 T2 TW T3
READY
ピィーッ

このWR、DBIN、ステータスを組み合わせて、コントロール信号をつくっています。INTA、OUT、INP、MR、MW、MW(SP)はWR、DBINのどちらかに同期したパルスになります。

室温16℃ きょうも雨か

シト シト シト

クロックを1μsecにすると、TK-80のクロックより2倍もありますから、マシンサイクルごとの待ちステートは必要ありません。

8080
10p
14
15
X-TAL 9MHZ
8224

TK-80キットの水晶発信器の周波数は18.432MHz、8224で1/9分周してますから、0.4882812μsecです。マシンサイクルごとにONE WAIT動作します。

0.4882812×2 μsec
T1 T2 TW T3 T1
WR
データ

マシンサイクルM2に移りましょう。

これは、MOV A,Mね。

DBINでHiZを解除OUTモードとなる

こうして、ああして、こうなって……説明はいりませんね。

マシン・サイクルM1
MOV A, M

こんなマンガありか?

7E
データバス
OUT
IN
ステータス
ROM
2073

私は＋1インクリメント

(PC)
2073
アドレス・バス
OUT
ロロロ
2074

ROMちゃんは、自分ではなにもできないのねえ。

フーンだ

T1 T2 T3 T1
φ1
φ2
SYNC
STSTB
CPUが読込む
データ・バス
TYPE①
7E
ステータスメモリ
TYPE①
アドレスバス
(PC)=2073
DBIN

CPUはステータスやコントロールを出しておいて時間がきたら勝手に割込む。ま、小さな独裁者だね。

私がRAMちゃんの読み出しタイミングを指示するの

うーんと次はHLペアの内容を出力するのね。

80
IN
データバス
OUT
21FF
アドレスバス
P.C.
A
B
H
2074
H·L
21 FF
DBIN
RAM

マシンサイクル M2
MOV A, M
M1
M2
T1 T2 T3 T1
φ1
φ2
SYNC
STSTB
CPUが読込む (A)←80
データバス 7E TYPE ② 80
ステータス メモリ TYPE ① TYPE ②
アドレス・バス (H)(L) = 21FF
DBIN

このメモリの番地指定をする時、チップセレクトはアドレスの上位ビットをデコードしました。

8080
アドレス

そこで、エートチップセレクトを書いたのは、とさがして、

135ページ…

ここで思いだしたのがチップセレクトするタイミングです。あの時はアドレスだけを問題にしてデータバスの内容については考えませんでした。

データ
CS
CS
LOW
'84 NEW ファッション

SYNCの間、データバスはメチャクチャです。

OUT S·I
データバス
OUT
80
チップセレクト
キャッ
アドレスバス
SYNC

ところがタイミング・チャートをよく見ると、アドレスが出力されている時でもSYNCが出ている時は、データバスにS·Iがのっています。

SYNC
データ・バス S.I.
アドレス・バス (H)·(L)

このチップセレクトというのはHiZの状態から出力状態に切り換えてデータバスを占有することなんです。

トライステート・ゲート
データ・バス
HiZ
ハイインピーダンス

8080
ROM #0
A2 A1 A0 CS
ROM #1
A2 A1 A0 CS
A0
A1
A2
A3
A4
A5
A B C
0 1 2 3 4
(MR)
DBIN
アドレス
データ S.I フローティング
DBIN
CS

RAMの時は、データが双方向性なので、もう一本、制御ビットが必要です。

そこで、このような混乱をさけるためステータス、DBIN、RWなどで作ったコントロール信号でうまくデータの交通整理をする必要があります。

RW
DBIN

ダイヤもCPUの命令ごとにキチンと決められてしまいます。DMA転送以外はCPUがバスラインの主導権を持っています。CPUはかしこいダダッ子なんです。

このバスは定員が決まっています。しかも、時間単位の貸切りバスです。一度に、2組以上の客(ビット)を乗せるようなダイヤを組んでは信用を落としますよ。

SYNC

ROM 2073番地
御一行様
D0～D8

8080のバスは2台です。コントロールを含めると3台ですが…。まあ2台です。

データは 8ビット乗り
D0～D7
アドレス 16ビット乗り
A0～A15
広告
路線 かつ 貸切りなので
ラッシュは ありません
8080バス 社長

平和への道はただひとつ、まず私たちユーザーが8080を理解しわがままを許すことです。これがハード設計の大切な所です。そして、それからは私たちの希望を聞いてもらいます。これがソフトプログラムです。

8080
ユーザーズ
マニュアル
(取説)

たとえ、プログラムされていなくても読み込んだものをOPコードと見なしてどんどん進行し、HALT命令コードにでも出くわさない限りストップしません。(FFFFになったら次は0000となる)

最初にRESETを入れると、プログラム・カウンタは0にリセットされ0番地のメモリの内容をOPコードとして(プログラムが)スタートします。

ムックリ
起きてちょうだい

ホールドはT2ステートで受け付けられます。READY要求との関連もあります。

T1 T2 T3
アドレス
データ
HOLD
READY
HOLDA

それは、総統にも寛大なお心があって、外部からのHOLD要求を受け付けられるのです。私はホールド受容のスポークスウーマンです。(ホールド・アクノリッジ)

あら、ホルダちゃん。

まず総統(8080)がおっしゃるにはデバイスの出力側がHiZ状態をとれなければバス利用会員とは認めないということです。CPUはデータ・アドレスバスともHiZ状態をとれます。

採用条件
バス

むずかしいことは、CPU内部でやっているので、コントロール信号はタイミングさえつかめばむずかしいものではありません。

このあと、
INTA
INP
OUT
について説明します。

私は、このことを外部に知らせ2つのバスラインを他の命令系統が使用してもよいことを知らせます。

DMA OK

こうすると、T3ステート以下HOLD要求が取り消されるまでCPUはTWステートを繰り返し、見かけ上停止し、データ・アドレスバスともHiZつまりフローティング状態になります。

T2 T3 T(4)
データ フローティング
アドレス フローティング
HOLDA

⑩ハード的なオペコード（割込命令（インタラプト））

8080はインタラプトという機能を持っています。どんなことに使われるかひとつ見てみましょう。

ここは鉄道の窓口

RST␣0

割込みポート

C7

CPU

メモリ

RST␣0～RST␣7はハード的にデータバスに挿入されるオペコードです。

と、そこにあたふたと一般の乗客がやって来ました。彼は担当者が仕事中なので、Aさんのあとに並びます。

A

B

担当者は決められた手順にしたがって周遊券づくりにとりかかります。

この人の担当業務

普通キップの発行

Mr. CPU

周遊券の発行

A

○×の周遊券をつくってください。

Mr. CPU

Mr. A

これに気付いた担当者は、こんなサインを出しました。

INT・E

普通切符の方はお申し出下さい 先にやります。係

B

さらにCさんがキップを買いに来ましたが、Bさんのあとに並ばねばなりません。

A

B

C

Bさんは電車が来る時間なのにキップが買えないのでハラハラしています。

A

B

このように、ある手順（プログラム）を実行中に別の手順が割り込み実行する。これがインタラプトの概念です。

おまちどうさま。

CPU

ありがとう。

A

そして、キップ客がいなくなると再び周遊券をつくる作業を開始するのです。

A

C

D

B

やれやれ、なんとか間に合いそうだ。

担当者はサービスに優先度をつけキップを買う客が来るたびに周遊券づくりを中断しまます。

CPU

INT

割込み許可を与えるのはソフト、つまりプログラムです。EI(Enable Interrupts)という命令を使います。マシン語は11111011B（2進表現）
F B H（16進表現）

プログラムリスト

INTEはインタラプト・イネーブルの略です。割込みが可能かどうか外部に知らせます。

発光ダイオード
8080
INTE
16
ドライバー

8080には、INTEとINTの2つの割込みのためのピンがあります。

INPUT
（割込みたのんます。）
INT
8080
OUTPUT
（割込み受け付けるよ。）
INTE

ハード的にコントロールできる8つのサブルーチンという感じです。

割込みプログラム
7 6 5 4 3 2 1 0
EI命令を入れて戻らないとこれ以後の割込みはできない。
必要ならばEI命令を入れて戻る。
独立したひとつのプログラムです。
必要ならばEI命令を入れて戻る。
EI命令を入れておく。
RET RST2 RET RST3 RET RST1
メインプログラム
←時間
スタート

この図は、CPUがある時間ひとつのプログラムしかやってないことを示しています。プログラムカウンタがひとつしかないので当りまえですね。

8080の場合割込みプログラムは8つまでとることができます。

私も働くの。
プログラムカウンタは私だけなのよ。もち、割込みプログラムも。
スタックポインタ

そして、そのプログラムの実行が終わるとリターン命令によって再びメインプログラムに帰るというものです。一度割込み状態になるとINTE状態が取り消されるので割込みプログラムの終りにEI命令を入れておくことが必要です。

EI命令
RET命令（リターン）
割込み3プログラム
中断
メイン
この図はプログラムの長さでなく実行時間を現わしています。

たとえば割込み3というボタンを押してやるとメインプログラムの実行が、一時中断されて、割込み3で指定したプログラムが実行されるのです。

割込み3
メイン
中断
時間
0 1 2 3 4 5 6 7

サブルーチンを呼びだすのはCALL命令でしたね。

メイン
サブルーチン
CALL
RETURN

割込み（サブルーチン）プログラムの場合、命令コードを1バイトにするために8つの先頭番地は固定されているんです。

(H)番地 メモリ（単位はバイト）
0000 ←0 8バイト
0008 ←1 8バイト
0010 ←2 8バイト

CALL命令コード
（3バイト命令）

CD B7 02

CALLされるサブルーチンの先頭番地、(02B7番地)ここにはOPコードが入っていなければなりません。

サブルーチンの場合、そのプログラムの先頭番地はCALL命令の中に含まれていましたね。

時間
プログラムカウンタ
OPコードのある番地
201A CD B7 02
201A CD
201B B7
201C 02
サブルーチンプログラム
02B7
02B8
201D
201E

0、1、2、3、4、5、6、7の8つの割込み番地は、3ビットでRST命令に挿入され、決定されます。

000 …… 0
001 …… 1
010 …… 2
011 …… 3
100 …… 4
101 …… 5
110 …… 6
111 …… 7

RSTn命令の一般的な表現は11NNN111です。このNNNに0～7の3ビットを入れると、それぞれのオペレーションコードができます。やってみましょう。

RSTn

このnは0～7までの数字。たとえば RST5

ね。

割込みレベル	2進表現のOPコード	16進のOPコード
0 …	11000111	(C7)
1 …	11001111	(CF)
2 …	11010111	(D7)
3 …	11011111	(DF)
4 …	11100111	(E7)
5 …	11101111	(EF)
6 …	11110111	(F7)
7 …	11111111	(FF)

NNN

この8つの1バイト命令は、今まで説明したプログラムメモリの中にはありません。別の所からデータバスにのせてCPUにフェッチさせます。

RSTnのOPコード

C7, CF, D7, DF
E7, EF, F7, FF

CPUはこのRSTn命令をフェッチすると、プログラムカウンタの内容をHとLの2バイトに分けて、おのおのスタックポインタをさしている場所に移します。

PC　H　L

RAM メモリ

L ←SP
H

そして、そのあと、プログラムカウンタの内容を0000 0000 00NN N000とするんです。このNNNは0～7を表わす3ビットです。たとえばNNN＝3とすれば、

0000000000011000

0　0　1　8

16進表現

0018番地が割込み3の最初のOPコードの場所になります。

8バイトおきに8レベル、合計64バイトがプログラムメモリの0000番地から003F番地にあることになります。

またよそ見して、しょうのない子。

割込み3

ん？

データバス

DF

番地　メモリ　電源リセットの時もここからスタート

0000　0
0008　1
0010　2
0018　3
0020　4
0028　5
0030　6
0038　7
003F

割込み3 このOPコードはDF (H) この結果プログラムカウンタは0018となります。ここに最初のOPコードを入れます。

0018

3FFA

H　L

40FE

40FE　L
40FF　H

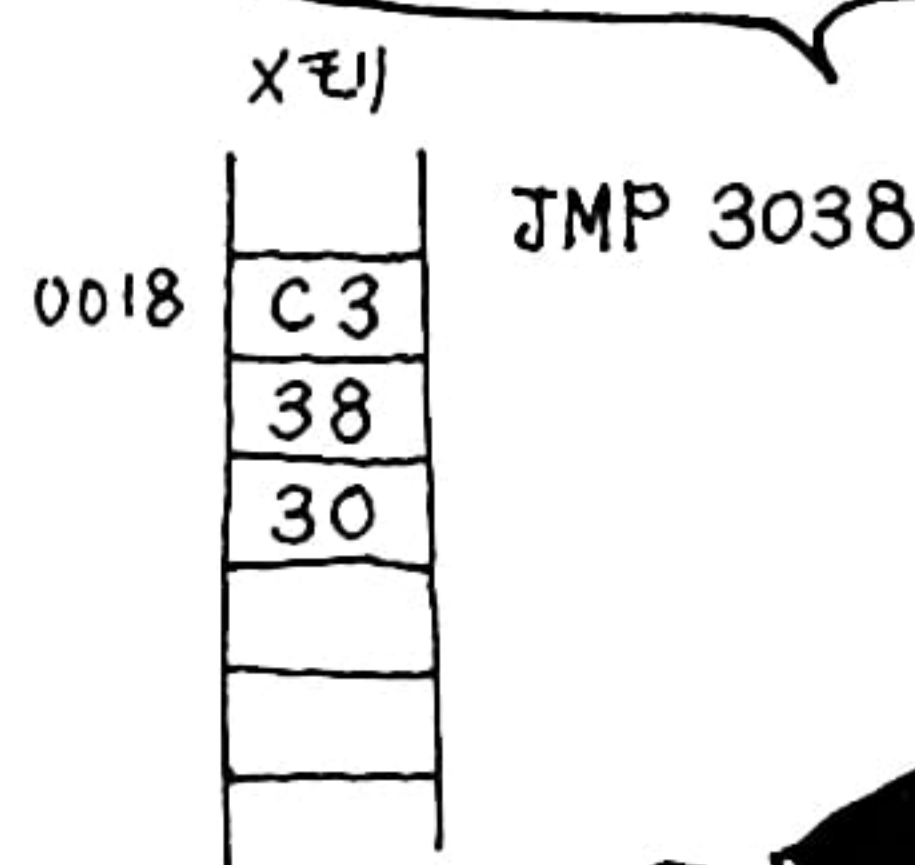
PC

SP

つまり0000～003Fまでは、普通のプログラムはメモリできません。また8バイトメモリでは割込み自身のプログラムもできないのでジャンプ命令で他の場所にとばします。

メモリ

0018　C3
38
30

JMP 3038

そして、とばした先に割込み3のプログラムを書いておけばいいのです。プログラムの終りにはリターン命令をつけてPCを元に戻します。

3038

割込み3のプログラム

C9 ←RET リターン命令

メインに戻る

割込みの信号は、ハードから任意の時刻にやってくるんです。それはリレーの接点とか、押釦の接点とかその他いろいろな機械的なもの、電子的なものです。

エンコーダ
押釦

各接点信号は優先順位をつけるエンコーダによって3ビットコードに変換されます。ここでNNNをつくるわけです。同時にCPUへのINT信号を出します。

3ビットコード
N
N
N
エンコーダ
INT
押釦を2つ以上に同時に押してもどれかひとつしか選ばれないようになってる。

では、次にRSTn命令をCPUにフェッチさせる方法についてお話しましょう。

RSTn命令をフェッチする ⑥
ここはトライステート・ゲートである CSにてHiZが解除される。
INTAの間RSTnを出力する
RSTn
接点をONする。①
+5V
データバス ⑤
INT
④ DBIN
8080
③ M1サイクル
ステータス ラッチ
INTA
割込みポート ラッチ
CS
チップセレクトで割込みポートを選ぶ
優先回路付 エンコーダ
2つ以上同時に押してもひとつしかコード化されないようにしておかなくてはならない。
② INTを出す（INTAがくるまで保持する。）

このブロック図に示された番号の意味を次のコマのタイミングチャートで説明しましょう。

ハードを読むためには時間と信号の関係を知る必要があります。

このINTがCPUに入力されCPUがINTE状態にあると次のマシンサイクルM1（OPコードフェッチサイクル）のステータスとしてINTAがデータバスに出力されます。

INT
8080
データバス 8Bit
ステータス ラッチ
INTA

このステータスのINTAと、CPUがデータを読み込むタイミングの出力DBINとのANDをとって、データバスにRSTn命令をのせるのです。

データバス
+5V
ラッチ（割込みポート）
エンコーダ
DBIN
INTA
CS
ステータス ラッチ
INT

8080のRSTn命令フェッチの方法と実行過程

（OPコードの読込み）

マシンサイクル　M1　M2　M3　M1

アドレスバスの内容（16ビット）　PC　?　SP-1　SP-2　W・Z

データバスの内容（8ビット）　S.I　RSTn　フローティング　S.I　PC(H)　S.I　PC(L)

SYNC（1ビット、ステータスが出てくることを知らせる。）　⑤　⑥ OPコードのフェッチ　メモリ　メモリ

STSTB（1ビット、ステータスをラッチするタイミング）

DBIN（CPUがデータバスの内容を読み込むタイミング。）　④

WR（外部メモリにデータバスの内容を書き込むタイミング）

INTE（EI命令を実行して割り込み可である状態）　③

INT（外部ハードからの割込み要求）　①②

ステータス（外部メモリで各々マシンサイクルのステータスをラッチする）　INTA, M1, WO　マシンサイクルの形 No.8　STACK　マシンサイクルの形 No.5　STACK　マシンサイクルの形 No.5

RSTすごろく

ふりだし

友子ちゃんの押したのは何番目の押釦？
CPUへは011番目というようにコード化して送る必要があるのです。

RSTのための押釦は８つが最大です。

エンコーダ

コード化することをエンコードといいます。

11NNN111

RST

11NNN111

OPコードレジスタ

データバスゲート開

DBIN
データバス イン

ステータス

INTA

M1

INT PORT

ポートという言葉も覚えてね。また出てくるよ。

割込みお願いしてもいいかしら？

INT

E I 命令実行済

INTE

マシンサイクルNo.8

ここからがRSTフェッチのマシンサイクルM1よ

受け付けたらお子す。

ホントはここに私つまりデータバスラッチがいるのよ。

OUT

IN

3F

データバス

3F

と

SP

41 00

インクリメント停止

3FFA

テキパキ

私はM2でPC(H)をメモリします。

データエリア

RAM

RAM

40FF
SP-1

40FE
(SP-2)

私はM3でPC(L)をメモリします。

アドレスバス

40FF

40FFをOUT

指示 次のアドレスはSPの内容

アドレス

0000 0000

1

00NNN000

1

私たちはテンポラリレジスタ、外部とは関係ないのです。（ソフトでさわれない）

W

Z

(W)

(Z)

00 18

これで割込み命令RSTn OPコードのフェッチとその実行が完了したのです。割込みプログラムがスタートするわけです。

これをPCに入れると 0000000000NNN000 となるわけ。

00 18

あがり

ただ、ここで私たちはポートという新しい言葉に出会いましたね。私たちはこれからCPU↔メモリから、さらに外の世界を広げていくのです。

8080の周辺デバイス(部品)として、プログラマブル割込みコントローラ8259というLSIがありますがここでは触れません。

これでだいたい割込みについての基本的な説明は終りです。ここではRSTn命令は、外部から割込みポートを通ってデータバスに入ってくると覚えてください。

これは、データバスがポートを通って外部と連絡しています。

これが、いままでの世界、CPUがやっていることは私たちにはわかりません。(目で確認できない)。

これぐらいで8080の割込みについての説明は終わりたいと思います。なおZ80などではもっと多彩な割込みができますので関心のある方は勉強してください。

この2つをまとめて書くとこうなります。データバスがようやくバス(共用車)らしくなってきたようです。

コントロールWAとINTAによって、それぞれのトライステートゲートを開ける時刻が違っているのでバスが混乱することはありません。

⑪入出力装置用ポート
(I/Oポート)

ウィィーン…
ウィィーン…

マイコンイコールパソコンと思い込んでいる人も多いのではないでしょうか。マイコンはこのようなかたちで工場の中でも活用されているのです。

マイクロコンピュータはソフトとハードを切り離して説明できないところがあります。IN命令、OUT命令もハードを抜きにしては説明しにくいのです。

8080やZ80には0000~FFFF(65Kバイト)の他に00~FFまでのI/O用ポートを指定できます。

I/Oポートへの8ビットデータの入出力は僕Aレジスタが窓口になります。

メインメモリ 65Kバイト

メモリ

256バイト

インプットポート

CPU

アウトプットポート

256バイト

8080の内部のお話

こんにちは。僕はAレジスタです。女ばっかしのこのマンガの中で貴重な存在。

Aレジスタ
またの名を
アキュムレータ
逆かな?

8080のINPUT(入力)命令と、OUTPUT(出力)命令の2つは、僕と外部デバイスとのやりとりなのです。

I/Oポート

Aレジスタ

Aラッチ

テンレジ

ALU

僕が一度に扱えるデータは8ビットです。I/O命令の時は各ビットに特別な意味を持たせます。

8Bit

D7 D6 D5 D4 D3 D2 D1 D0

IN、OUT 2つのポートを構成している部品はデバイスといい、メモリの番地と区別するためデバイスNo.と言ったりします。

I/Oポート　デバイスNO.　8Bit ラッチ

メモリ　アドレス　8Bit メモリ

メモリの番地に相当するポートのデバイスNo.は命令の2バイト目に書いておきます。IN、OUTの2つの命令は2バイト命令です。

INPUT命令　11011011…DB(H)

ここに INPUT ポートのデバイスNo.をかく

OUT PUT命令　11010011…D3(H)

ここに OUTPUT ポートのデバイスNo.をかく

1バイト(8Bit)では0~255(D)の256種のポートを指定できることになります。

256 { 0000 0000…00(H)
1111 1111…FF(H)

FF

でも、ふつうの制御にマイコンを使用する場合、そんなにたくさんのINPUTやOUTPUTが必要なわけではありません。だってボタンなら2,048個分なんですもの。

256×8Bit
=2,048

I/OにもI/O表、つまり、デバイスNo.に割り当てた信号名を記入したものが必要です。

メモリにはその内容を示すアドレスマップが必要でしたね。

Aレジスタから出力する内容

OUT PUT表(IN PUT表)

命令の、2バイト目で指定する。

DIV No.	D7	D6	D5	D4	D3	D2	D1	D0
00								
01								
02								
03								

では、まずインプット命令からお話しましょう。ここでは INPUT・インプット・入力はそれぞれ同じ意味で使っていますから固く考えないでください。その時の気分で使い分けているだけですから。

INPUT
インプット
入力

マイコンへのインプットとしては制御のためのボタンとかリレー接点とかリミットスイッチとがあります。また計測データなどはBCDコードでインプットします。

外部入力

インプット・ポート

PB1 (押釦)
PB2
R1 (リレー)
R2
TR1 (タイマー)

測定器
押釦
キーボード
ディジタルスイッチ

作る製品の仕様が決まったら、入力するものも決まりますからINPUT表をつくります。

インプット機器

プリント押釦

A B C D COM

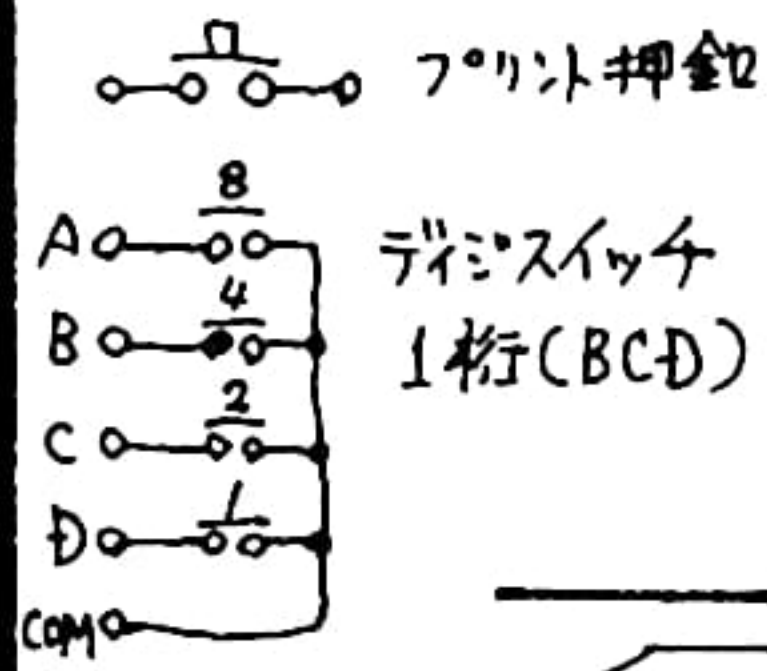
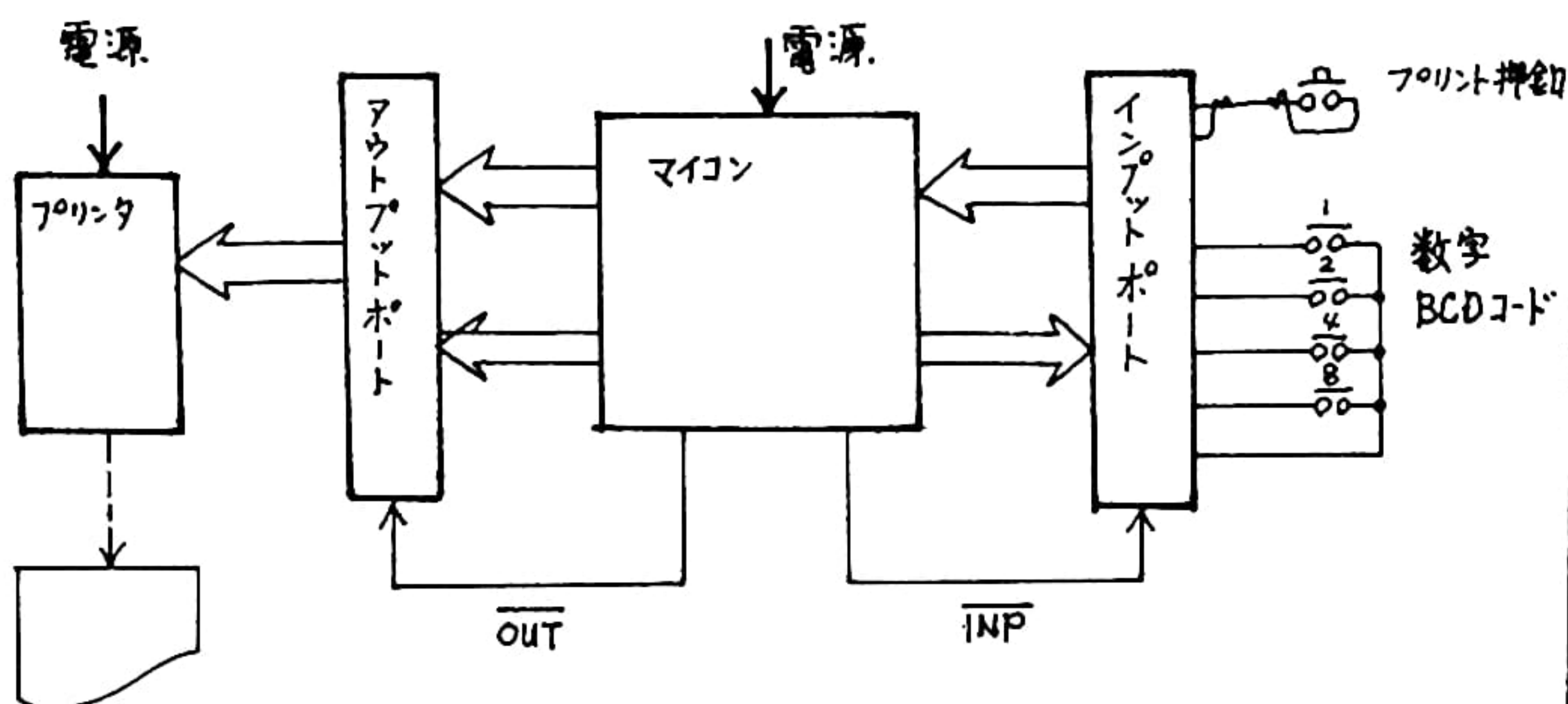
8 4 2 1

ディジスイッチ 1桁(BCD)

たとえば、D0～D3にディジスイッチのデータを、

D7にプリントの押ボタンを割り当てそれのデバイスNo.を○○としました。

INPUT表

DIV No.	D7	D6	D5	D4	D3	D2	D1	D0
00	プリント押釦				ディジ 8	ジ 4	スイッ 2	チ 1
01								

数字1桁をマイコンを通してプリントすることでI/Oを勉強しましょう。

マイコン

ディジスイッチ

プリント押釦

プリント

I/OとマイコンのBの関係を示すブロック図

電源

プリンタ

アウトプットポート

電源

マイコン

インプットポート

プリント押釦

数字 BCDコード

1 2 4 8

$\overline{OUT}$

$\overline{INP}$

インプットの場合、このポートはデータバスに出力する側になります。だから出力がHiZの状態のとれるICを使わなければなりません。

BUS

オープンコレクターのNANDゲートでもいいんですけど…

コントロール

D0 D1 D2 D3 D4 D5 D6 D7

データバス

← 入力

オープンコレクターの場合は、積分回路になるので応答速度が遅いんです。

データ Bit

5K

Dn

コントロール

回路が長いと

遅れ

そこで、やっぱりトライステート・ゲート、これは、オープンコレクターの欠点改良のために考えられたものです。

D1 D2 D3 D4 D5 D6 D7 D8

入力

コントロール

このコントロールは、やはりCPUからのステータスと読込みタイミングのDBINを使います。

コントロール・バスにinpという1Bitの線(信号)が出ていますね。このポートのセレクトにはこのINPを使います。

DBIN

INP

$\overline{INP}$

コントロール HOLDA

データバス
$\overline{INP}$
エンコーダ
アドレス A2 A1 A0
C B A
1 2 3 4
00
01
02

でも、これは1ポート・8Bitだけです。増設するためにはアドレスバスに出力されるデバイスNo.を使ってデバイス・セレクトをします。

ポートの増設

これで8ビットのインプットポートがひとつできました。

$\overline{INP}$

マシンサイクルM1
（命令コードフェッチサイクル）

次のデバイスNo.を読み込みましょう。

DB…これはINPUT命令ね。

DB
IN
ROM
データバス
2011
PC
+1
アドレス・バス

INPUT命令のフェッチとその実行の過程を見てください。まずOPコードとしてINPUT命令DBをフェッチします。この命令は2バイト命令です から さらに2バイト目をフェッチします。

2011 DB
2012
2013

マシンサイクルM2
（命令コードの2バイト目を読込み）

00
W
Z
DB
ROM
データバス
2012
2011
(PC)
アドレスバス

マシンサイクルM2では、2バイト目を読み込むとともにそのデータをW・Zのレジスタに入れます。

マシンサイクルとは何だったかな？ 復習したい方は146ページに戻ってください。

マシンサイクルM3

うーん、ひとつのデータを読むのにも大変なことやってるんだなあ。

INPUT ポート デバイスNo.00の内容
A
1
データバス
インプットポート
外部
W Z
0000
指示 次のアドレス (W)(Z)
$\overline{INP}$
デバイスNo.00
アドレス・バス

マシンサイクルM3でW・Zの内容でアドレスバスに出力されるので、これでデバイスNo.の指定ができます。コントロール信号$\overline{INP}$で指定したデバイスNo.のデータをAレジスタに読み込みます。

INPUTポートのデバイスNo.00に割り当てたプリントボタンのビットはD7です。

DIV No.	D7	D6	D5	D4	D3
00	プリント				

このビットを調べる

大ざっぱに書くとこんなフローチャートになります。

INPUT
プリント押釦は押されたか？
NO
?
YES
OUTPUT

ところで、プリントボタンを押したかどうかCPUがどうして判断するんでしょうか。これはソフトの問題です。

これは、2バイト命令で2バイト目のデータとAレジスタの各ビットごとのANDをとるという命令です。

テンポラリレジスタ
8 0
10000000
E6
80
結果をもどす。
00000110
A
00000000
ゼロフラグ

73ページの命令一覧表を見てください。この下の方にANIというのがありますね。

E 6
11100110 <B1> … E6
<B2> …
プログラムの時決める
〈実行〉
(A)← (A)∧(B2)
(c)← 0
∧…論理積

ここに'1'がたったかどうか見るためにはAレジスタに読み込んだデバイスNo.00のデータと、10000000、つまり80HとのANDをとってその結果が0かどうかの判断をすればいいんです。

ソフトによるANDの方法

つまりボタンが押されるまで、ゼロのフラグは上がりっぱなしなので、ここにジャンプ命令を入れて下げた時に通過するようにすればいいのです。

ポートデバイスNo.00
"D7"
AND
80
プログラム
ZERO

そして、その結果が0か、そうでないかで、フラグが変化するのです。ここが、ソフトで判断できるポイントなのです。

この命令は8つのANDゲートをソフトで実現することができるのです。

まだ他にもAレジスタの特定のビットを調べる方法はありますが、要は、そのビットを操作してフラグを働かせ、そのフラグを条件にしたジャンプ命令を使うことなんです。

これがCPUに制御信号をプログラムでフェッチさせる方法です。割込み命令と、どこが違うか自分で確かめてみてください。

プログラムでハードの変化をキャッチする

ANDをとって、条件ジャンプさせるわけですね。

JZ…ゼロならジャンプ、Z=1ならばゼロ。くそ、ややこしいナ。

INPUT
00
ANI
80
JZ
Z=1
Z=0
押されるまでグルグル回っていることになる。
プリント押釦が押されたと判断

私が今ボタンを押しましたので、プログラムは、このループから抜けることになります。

INPUT
00
ANI
80
JZ
Z=1
Z=0
プリント

では簡単な例をひとつ。印字の押釦を押すと、ディジスイッチの数字1桁をプリントするものです。

チョン

さて、次は OUT命令です。CPUからデータを出力するのです。

〈B1〉 OUT コード
〈B2〉
OUT

このようにして、CPUはプリントボタンが押されたと判断して次のステップへ進むわけです。コンピュータに何をどのように判断させるかは、人間のつくるプログラムにかかっているわけです。

前ページではANI命令実行の結果Aレジスタはゼロになるので、Z＝1、この場合は1となり（注1）Z＝0なのです。

Z=1なら指定された番地にジャンプ
JZ
Z=0なら次に進む

注1
1となるのはD7ビットだけ、Aレジスタの内容は80となる

前ページのとどこが違うか当ててください。

ANI命令の2バイト目
8 0
1 0 0 0 0 0 0 0
INPUT PORTからのデータ
プリント押釦
データ
1 0 0 0 0 1 1 0
実行結果
1 0 0 0 0 0 0 0
ゼロ・フラグ"0"となる

そして、OUTの場合、出力するビットのタイミングが必要な場合がありますからタイミングチャートを書いてからプログラムするようにします。

プリントデータ 4Bit
40msec（ソフトタイマー）
プリント指令 P/C
約40msec ソフトタイマー

番号はINPUTと同じですがコントロール信号のINP OUTは同時に出ることはないからOKです。

あっ、そうそう。どこに何を出力するのかOUTPUT表をつくっておかなくちゃいけませんね。

OUT PUT表

DEV No	D7	D6	D5	D4	D3	D2	D1	D0
00					プリンタ 8	4	2	1
01	プリント指令							

アウトプット・ポートはフリップ・フロップでラッチするのが基本的なやり方です。この図はDEV.No.00の8ビットラッチです。

OUT
データバス
D0
D1
D2
D3
D4
D5
D6
D7
4Bit ラッチ -175
4Bit ラッチ -175
ドライバー（通常オープンコレクタ）
基板外回路へ
DEV.No.選択（アドレスバス）
A0
A1
A2
デコーダ
00
データバス
OUT

フリップ・フロップの出力側は、そのまま基板の外へ出すことはできない。ノイズで信号が反転することがあるからです。

これで、Aレジスタの値をインプットポートDEV.No.00の8ビットと同じにしたのです。

1 0 0 0 0 1 1 0

A

```
2011  DB 00      LOOP IN 00
2013  E6 80           ANI 80
2015  CA 11 20        JZ LOOP
2018  DB 00           IN 00
```

ちょっとプログラムしてみました。

アウトプットのOPコードはD3ですからDEV.No.00に出力するためには、D3 00とすればいいですね。でも、Aレジスタの内容はANI命令を実行して80となり、インプットポートから読み込んだデータではありません。もう一度データをAレジスタに読み込んでおかなければなりません。

OUT命令を使う時は、その時、僕つまりAレジスタの内容がどうなってるかよーく考えてからにしてね。

アウトプットポート
DEV.No.00

D0 D1 D2 D3 D4 D5 D6 D7
ラッチ F.F.
0 1 1 0 0 0 0 1
データバス
OUT
W Z
0 0 0 0
アドレスバス

これをアウトプットポートのDEV.No.00に出力すればいいですね。OUT命令はIN命令とよく似ていますから、M1、M2サイクルを省いてM3だけを示します。感じをつかんでください。

```
201A  D3 00      OUT 00
```

データ
プリント指令 P/C

P/Cはもう少しあとから出力したいのです。

今、時間にするとこんな所にいるんですよ。

プリントキバン
ラッチ
D0 (1)
D1 (2)
D2 (4)
D3 (8)
コネクター
コネクター
プリンター
4Bit BCDコード1桁

OUTPORT DEV.No.00の下位4BitはBCDコード1桁としてさらにプリンターに接続されます。

うーん、およそ2000倍もの時間をどうしたらいいんでしょう。マイコンは早すぎるので時間の調整をしたい時がたびたび出てきます。そこで、ソフトによるタイマーのプログラムを考えてみましょう。

$$\frac{40\,\text{msec} \times 1000}{20} = 2000$$

```
① 201A  D3 00    OUT 00
② 201C  3E 80    MVI A 80
③ 201E  D3 01    OUT 01
```

なーんと、このままだと①から③までの時間差は約20μsecなのです。(クロックを1μsecとしてのお話)

① ③
1μsec

P/CはDEV.No.01のD8と決めてありますから80(H)をAレジスタに入れ、それをOUTすればいいんですけど時間的に40msecずらせてやりたいのです。

OUTPUT表より

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0
8				0			

時間としてのクロック数を設定する。N1
クロックを0からカウントする。N2=N1+1
NO
N1=N2？
YES
えーと、どうしたらいいのかな？

前にプログラムを実行する時間を計算しましたね。この実行にかかる時間をうまく使ってタイマーにするんです。
1μsec＝1ステート
φ1
φ2

マイコンをはじめ、すべてのコンピュータはそれぞれの持つクロックを最小単位として働いているのです。
φ
XTAL 9MHz
8080
1/9分周
8224
φ1
φ2

そこで上のフローチャートをちょっと書きかえてみます。
ある数をレジスタrに入れる。
レジスタrから1をひく
NO
フラグは0か
YES
わざと、ある回数だけループさせてタイマーとしての時間かせぎをするわけ。

それは、ゼロフラグを使っての条件ジャンプが、便利だからです。レジスタにある数字から1ずつ引いていけば、いつかはそのレジスタはゼロになりますね。その時にゼロのフラグが変化するわけです。
ゼロ

人の場合は、あと何時間たったらという具合に時間の増加の方向で考えるのが自然ですね。でも計算機の場合は減算の方向で考えてプログラムします。

ニーモニック	DCR␣r
OPコード	00DDD101
サイクル数	1
ステート数	5
オペレーション	(r)←(r)-1

レジスタrの内容を－1する。キャリーを除くすべてのフラグは影響を受ける。

レジスタr
A 111, B 000, C 001, D 010, E 011, H 100, L 101
パリティ ゼロ サイン

前に紹介したDCR rという命令が、それです。
DCR Aというぐあいに7つのDCR命令があります。OPコードをつくってみてください。

このレジスタから1をひくという私たちの常識からすれば変な感じのすることもコンピュータとしては当り前のジョーシキなのです。この-1をデクレメントといい、DCR命令と略しているのです。

Bレジスタには01を入れましたからループすることなく通り抜けますね。この時間がタイマーとしての最小単位となるわけです。1ステート1μsecとすれば17μsecです。

```
      MVI␣B, 01
TIME  DCR␣B
      JNZ␣TIME
```

MVI B,01
DCR B 5ステート
JNZ 10ステート
ソフトによるタイマープログラムの一例

レジスタはBレジスタを使いましょう。
さて

そして、DCR命令後実行のレジスタの内容は、フラグの'1'か'0'で条件ジャンプを使って調べることができます。ゼロでない時DCRをくり返せばいいのでこの場合はJNZ命令を使うことになります。
JNZ

ずらせる時間なんてだいたいでいいのですから17≒20として、2,000回ぐるぐる回してやりましょう。キチンとしたことが好きな人は17として計算してください。

そこでMVI B,2000(D)となります。でも、ハンドアセンブルする時は、これは16進(H)に変換しなければなりません。

ちょっとサービスして10進→16進への変換方法をお教えしておきましょう。まず、10進→2進の変換をします。

2000(D)→ ? (B)

デシマル
(10進数の意)

バイナリー
(2進数の意)

これは10進の数字を順次2で割っていくことで得られます。まず2000を2で割ります。答は1000で余りはありませんので0とします。これは答の横にメモしておきます。

2)2000
1000 …… 0

1回目の答

余り

この要領で次々と割っていき余りをメモしていきます。

2)2000
2)1000 …… 0
2) 500 …… 0
2) 250 …… 0
2) 125 …… 0
62 …… 1

ここで初めて余りが1と出ました。続けます。

2)2000
2)1000 …… 0 a
2) 500 …… 0 b
2) 250 …… 0 c
2) 125 …… 0 d
2) 62 …… 1 e
2) 31 …… 0 f
2) 15 …… 1 g
2) 7 …… 1 h
2) 3 …… 1 i
1 …… 1 j
1 k

順序を示すためにつけたもので他の意味はありません。

もう割れないのでこれで終りです

k j i h g f e d c b a
1 1 1 1 1 0 1 0 0 0 0

これをkの方から順に並べると2000(D)を11111010000(B)に変換したことになるんです。

2進→16進の変換はもっとかんたん。4ビットずつ区切って16進コードに直せばいいんです。

11111010000(B)
7 D 0 (H)

つまり…

2000(D)→7D0(H)となります。

検算
7×16² = 1,792
+) D→つまり13×16 = 208
2000(D) ←……あってる!

MVI B, 7D0 ?!

そんじゃま…

と?

えへ、これは失敗です。Bレジスタは8ビット、つまり256(D)しか扱えないのでありましたっけ。

MVI B, FF(H)
↓
255(D)
が限界なの

では、2000=40×50というぐあいに分割してウェイト(待ち)時間のルーチンを考えていきましょう。

ポイント → ×

40

50

レジスタの内容
1 MVI B, 02 (B=2)
2 MVI C, 02 (C=2)
3 DCR C (C=2-1=1)
4 JNZ T1 (C≠0) T1にジャンプする
5 DCR C (C=1-1=0)
6 JNZ T1 (C=0) T1へジャンプしないで次へ
7 DCR B (B=2-1=1)
8 JNZ T2 (B≠0)
9 MVI C,02 (C=02)
10 DCR C (C=2-1=1)
11 JNZ T1 (C≠0)
12 DCR C (C=C-1=0)
13 JNZ T1 (C=0)
14 DCR B (B=1-1=0)
15 JNZ (B=0) ここで抜ける。
前のコマのプログラムをCPUが実行すると、こんなぐあいになるんです。
まず最初わかりやすいようにB、Cそれぞれに2を入れた時どうなるか、一般的に成立する式を導いてそれから2000=40×50の場合を考えることにします。フローチャートをプログラムすると……
MVI B, 02H
T2:MVI C, 02H
T1:DCR C
JNZ T1
DCR B
JNZ T2
B、C2つのレジスタを使って2段構えにしました。
MVI B
MVI C
DCR C
C≠0
JNZ
C=0
DCR B
B≠0
JNZ
B=0
WAIT ルーチン (タイマー)
まず、フローチャートを書こうね。
これを見て関係式を導きます。
Bレジスタの内容
Cレジスタの内容
DCR JNZ }…(B)×((C)+1)=2×3=6回
MVI B, 02 … =1回
MVI C, 02 … (B) =2回
ホントかなあ、と思う人はB=3、C=2またはB=2、C=3として演習問題のつもりでやってください。
DCR JNZ } の組は17ステート、MVI、r、D8の組は7ステートです。そこで、1ステート1μsecとすると、
(2040×17+40×7+1×7)×1μsec=34967
=34967μsec .
となります。目標は40msec、これは35msecですからだいたいいい線でしょう。
これじゃ気持の悪い人はB、Cにもっと40msecに近くなるものを入れてくださいね。
関係式ができましたので、B=40、C=50を入れて時間を計算することにします。
実行回数
DCR JNZ }…(40)×(50+1)=2040回
MVI B, 40(D) =1回
MVI C, 50(D) …(B)=40回
Bレジスタの内容
サブルーチンのラベル (最初のOPコードの番地)
WAIT: MVI B, 28H
T2: MVI C, 32H
T1: DCR C
JNZ T1
DCRB
JNZ T2
RET
リターン文をつける。これは10ステート必要。気になる人はこれも含めてください。34,967+10=34,977μsec
これで35ミリ秒のタイマーができたことになります。何度も使えるようにサブルーチンにしておきましょう。
2)40
2)20 …0
2)10 …0
2)5 …0
2)2 …1
1 …0
2)50
2)25 …0 a
2)12 …1 b
2)6 …0 c
2)3 …0 d
1 …1 e
f
40と50を16進に直しておきましょう。
101000
2 8
f e d c b a
110010
3 2
∴ 40(D)=28(H)
50(D)=32(H)

私だったらタイム・チャートの方を修正しちゃいます。

データ　~~40~~ 35 △

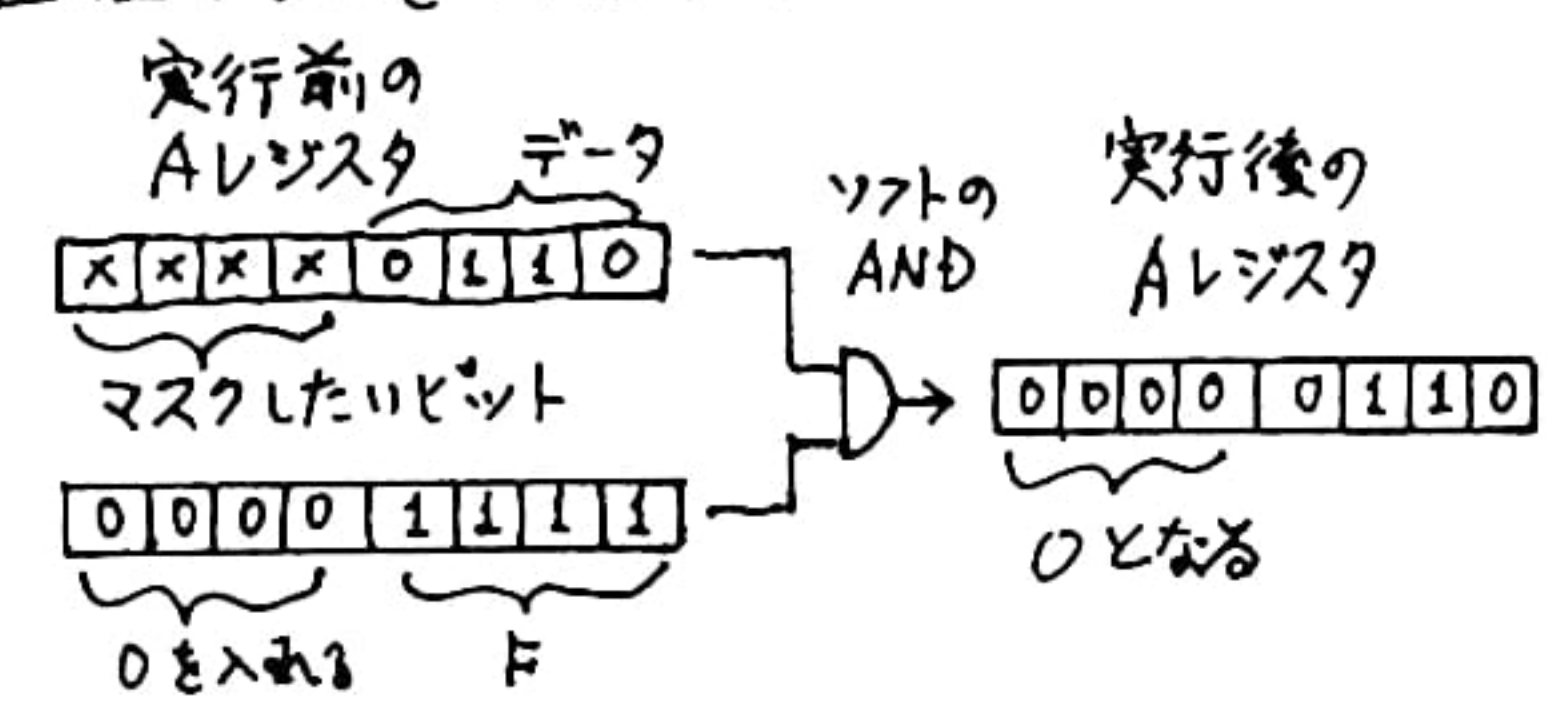
P/C

~~40~~ 35msec △

△…1回訂正した、の意

```
        ⋮
Loop:  IN ␣00      ; 00ポートよりデータ読込み
       ANI ␣80     ; } P/C(プリントコマンド)押釦のチェック
       JZ ␣Loop    ; }   押されるまでここでグルグル回っている。
       IN ␣00        00ポートのデータを改めて読込み
       ANI ␣0F       上位4Bitをマスクする。

       OUT ␣00       データをOUT PUT
       CALL ␣WAIT    35 msec 待つ
       MVI ␣A, 80    P/C ビット(D7)をセットする
       OUT ␣02       P/C を OUT PUT
       CALL ␣WAIT    35msec 待つ
       MVI ␣A, 00    P/Cのリセットをセット
       OUT ␣02       P/Cをリセット
       CALL ␣WAIT    } 35×3=105 msec +α
       CALL ␣WAIT    }   待つ
       CALL ␣WAIT    }
       OUT ␣00       データをリセット
                       (Aレジスタは今00が入っている)
        ⋮

WAIT:  PUSH ␣B
       MVI ␣B, 28H   35msec WAITの
T2:    MVI ␣C, 32H   サブルーチン
T1:    DCR ␣C
       JNZ ␣T1
       DCR ␣B
       JNZ ␣T2
       POP ␣B
       RET
        ⋮
```

メインプログラム

サブルーチン

実行前のAレジスタ　データ

×	×	×	×	0	1	1	0

マスクしたいビット

0	0	0	0	1	1	1	1

0を入れる　F

ソフトのAND

実行後のAレジスタ

0	0	0	0	0	1	1	0

0となる

<問題>

LOOP → 2000 番地

WAIT → 2100 〃

として、ハンドアセンブルし、プログラム リストを作成してください。

入出力のプログラムを整理すると、このようになります。

自分でモデルを想定して書いてみると、よくわかりますよ。とかく、人の書いたのは、先入観もあってよくわからないものです。

マスクという言葉の感じをつかんでください。

HOLDA

$\overline{INTA}$

$\overline{OUT}$

$\overline{INP}$

$\overline{MR}$

$\overline{MW}$

$\overline{MW(SP)}$

I/Oポートのコントロール信号

74LS368

I/Oポートのコントロールはステータスインフォメーション(S.I.)のデータバスのビットD4、D6を使います。これでコントロール・バスのすべてが明らかになりました。

D6　INP　D4　OUT

これでディジスイッチの数字1桁をCPUを通してプリントするシステムができたわけです。

マイコンキット(制御)

(OUT PUT)プリンタ

ディジスイッチ

押釦(インプット)

6

ここに入るプログラムによってI/Oポートの働きが決定されます。
0000
メインメモリ
FFFF
インプットポート
アウトプットポート
デイジスイッチ
1
2
4
8
BCDコードでたとえば6と設定
押釦「印字」
D0
D1
D2
D3
D4
D5
D6
D7
デバイスNo.0
ラッチ
ラッチ
プリンタ
P/C
6
デバイス NO.0
HiZ ゲート
CPU 8080
データ
アドレス
ステータス
INP
OUT
データバス
HiZ ゲート
ラッチ
ラッチ
アウトプット側デバイス No.1
この例題のI/Oポートの回路はだいたいこんな感じです。
D0
D1
D2
D3
D4
D5
D6
D7
インプット側デバイス No.1
デコーダ
0
1
2
3
A0
A1
A2
A3
デコーダ
0
1
2
3
IN命令実行時のデバイスNo.選択
OUT命令実行時のデバイスNo.選択
アドレスバス（メインメモリへ）

制御用I/Oはともかく、大量のデータを メインメモリ から アクセスしたい時には、INやOUTの命令を使って1バイトずつ扱っていたのでは実用上問題になることがあります。
FA
OA

そこで考えられるのがハード的に回路を切り換えてメモリアクセス専用にして時間を短縮する方法です。
これがDMA転送といわれるメモリアクセス方式です。8080ではHOLDとHOLDAのピンでこれを行ないます。
DMAはデータの宅急便
僕なら1バイトごとにI/O命令を使う。
ファイル
データ
メモリ

CPUへのHOLD信号がくると、CPUはHOLDA信号を出し、自分が専有していた3つのHiZゲートの出力をフローティングにします。これでシステムバスがCPUから解放されたことになります。
HOLD
HOLDA
データ バス
アドレス バス
コントロール バス

このバスの解放はHOLDがハード的に解放されるまで続きますのでその間CPUのプログラム実行はストップしているわけです。
DMA
システム・バス
メモリ
CPU

HOLD信号をCPUに送るのはDMAのコントローラです。DMA コントローラは メモリアクセスの情報をあらかじめCPUから受け取っているはずです。
メモリちゃんと交際させて
DMA
HOLD
システム・バス
メモリ

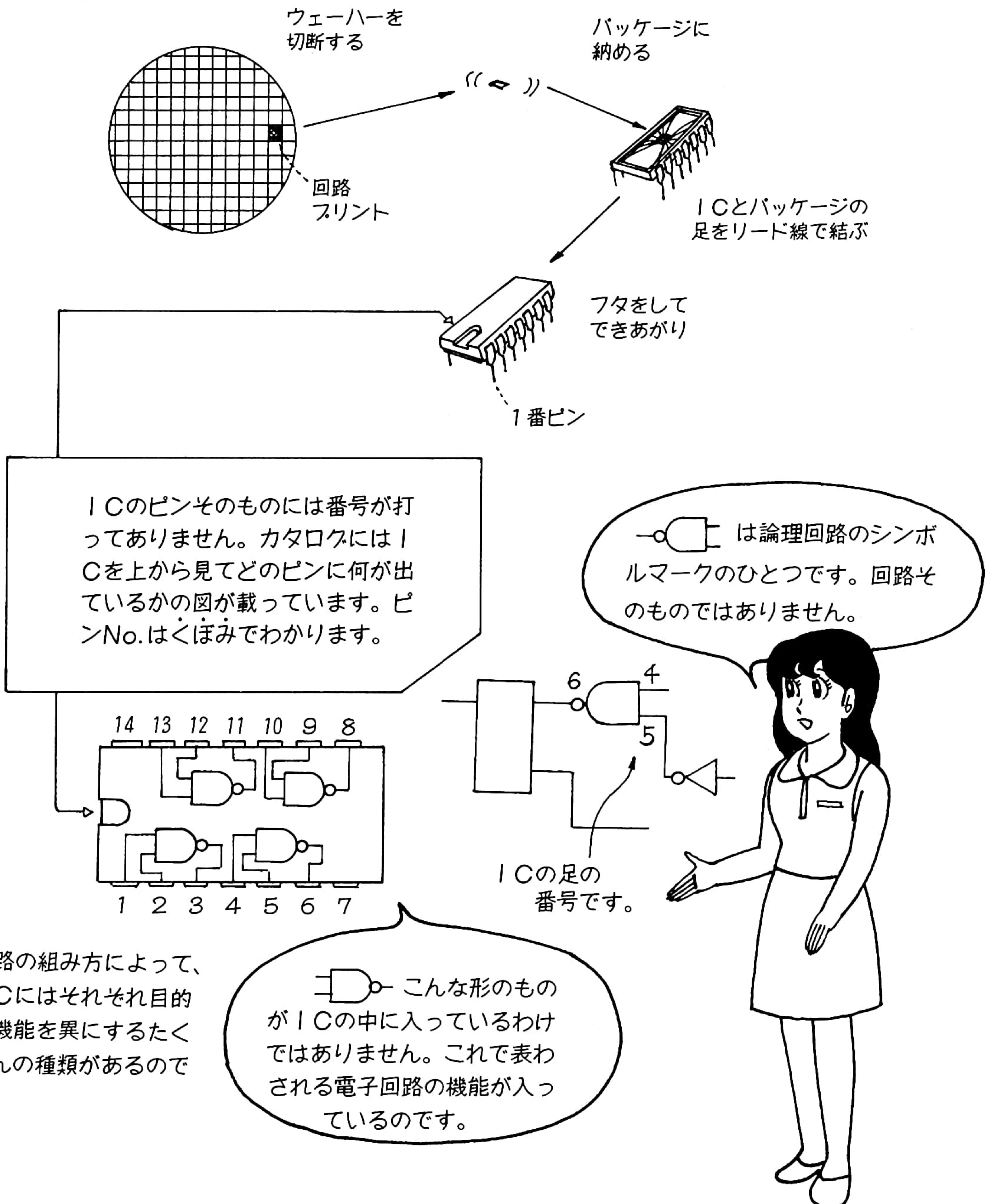
つまり、プログラムの実行の中でDMAの処理ルーチンが現われるとCPUからDMAのコントローラに転送情報としてのデータを送ります。コントローラはこのデータをもとにしてDMAを……。
ホンヤク
LOAD "TEST"
DMAルーチンへ入れ
カタ カタ

DMAが完了するとDMA側がバスを解放し、HOLD要求を止めます。そして、再びCPUがメモリを占有してプログラムが走るというシステムです。
LOAD "TEST"
OK

フロッピーディスク付きのパソコンなどではこのDMA方式を採用しているものがあります。
DMA 機能付

8080の説明はこれぐらいで終わりたいと思います。最後に8080の占領範囲を図示してみました。
8080
メモリ

ＩＣのQ&A

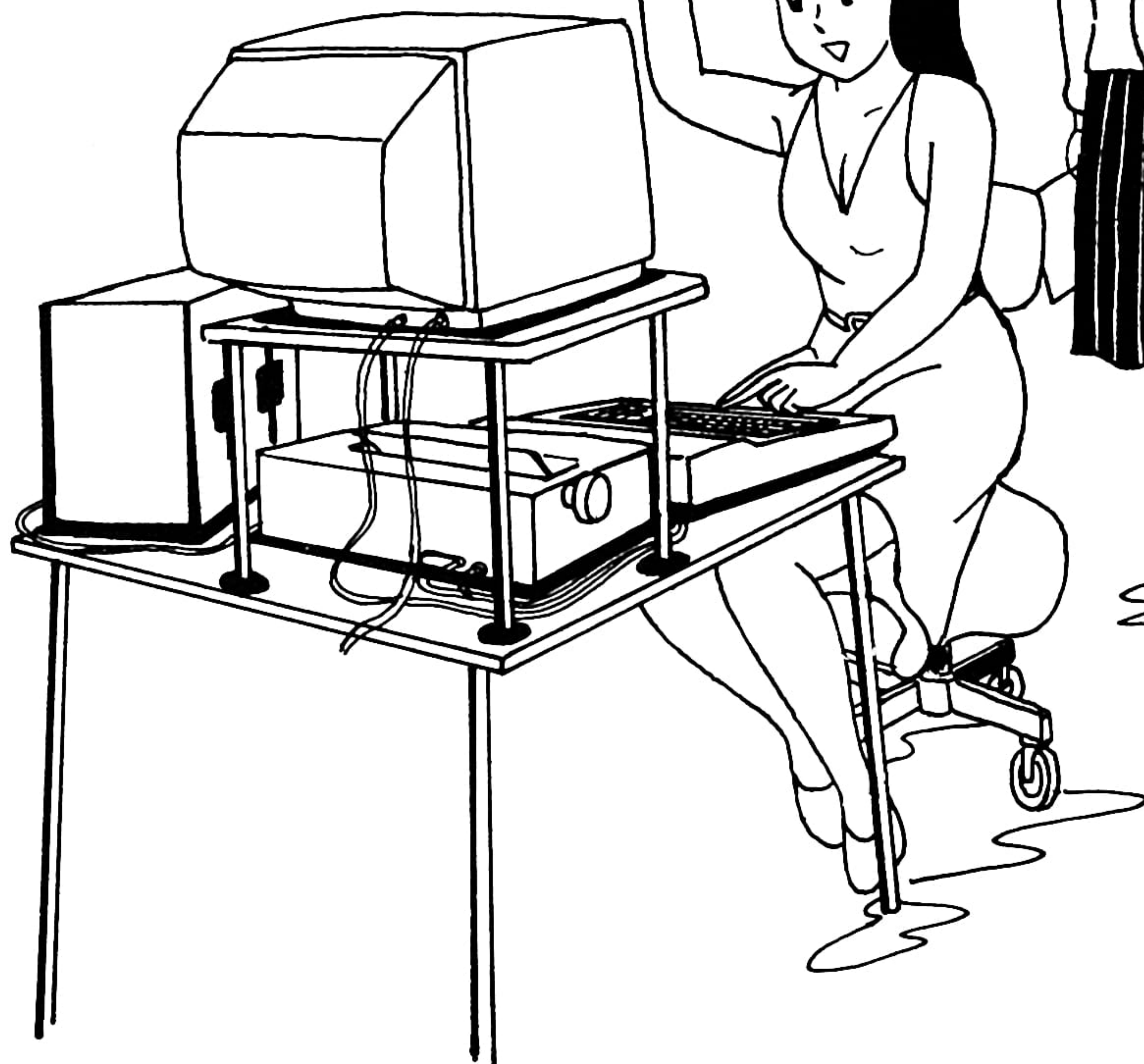
＊ＩＣについてもう少し話してください
　ＩＣというのは一般にムカデのような形をしていますが本当はあれはＩＣを入れるプラスチックのケースなのです。そしてムカデのような足はプリントキバンやソケットに入れやすいようになっているのです。ＩＣ自身は小さくて薄いものです。ＩＣはせんべいのような丸いウェーハーにたくさんの同一回路を〝写真印刷〟して作られます。そして切断してひとつひとつの回路に分けパッケージの中に納めるのです。

D CP/Mとアセンブラの活用知識

BASICを走らせるためには、BASICインタプリタが必要です。
ニーモニックで書いたプログラムを走らせるためにはアセンブラが必要です。
では、このインタプリタやアセンブラを作るためにはどうしたらいいのでしょうか。
このようなプログラムを作ることをソフト開発と呼んでいますが、このソフト開発用に作られたのが、ディスクオペレーティングシステムを持つCP/Mという8080用のソフト開発システムなのです。
CP/Mは8ビットパソコンのOSとしてオールRAM型パソコンに採用されつつあります。

①CP/Mとは何か

日本はいろんな工業製品をアメリカなどに輸出していますが、パソコンではボツボツ輸出しているという程度なのです。

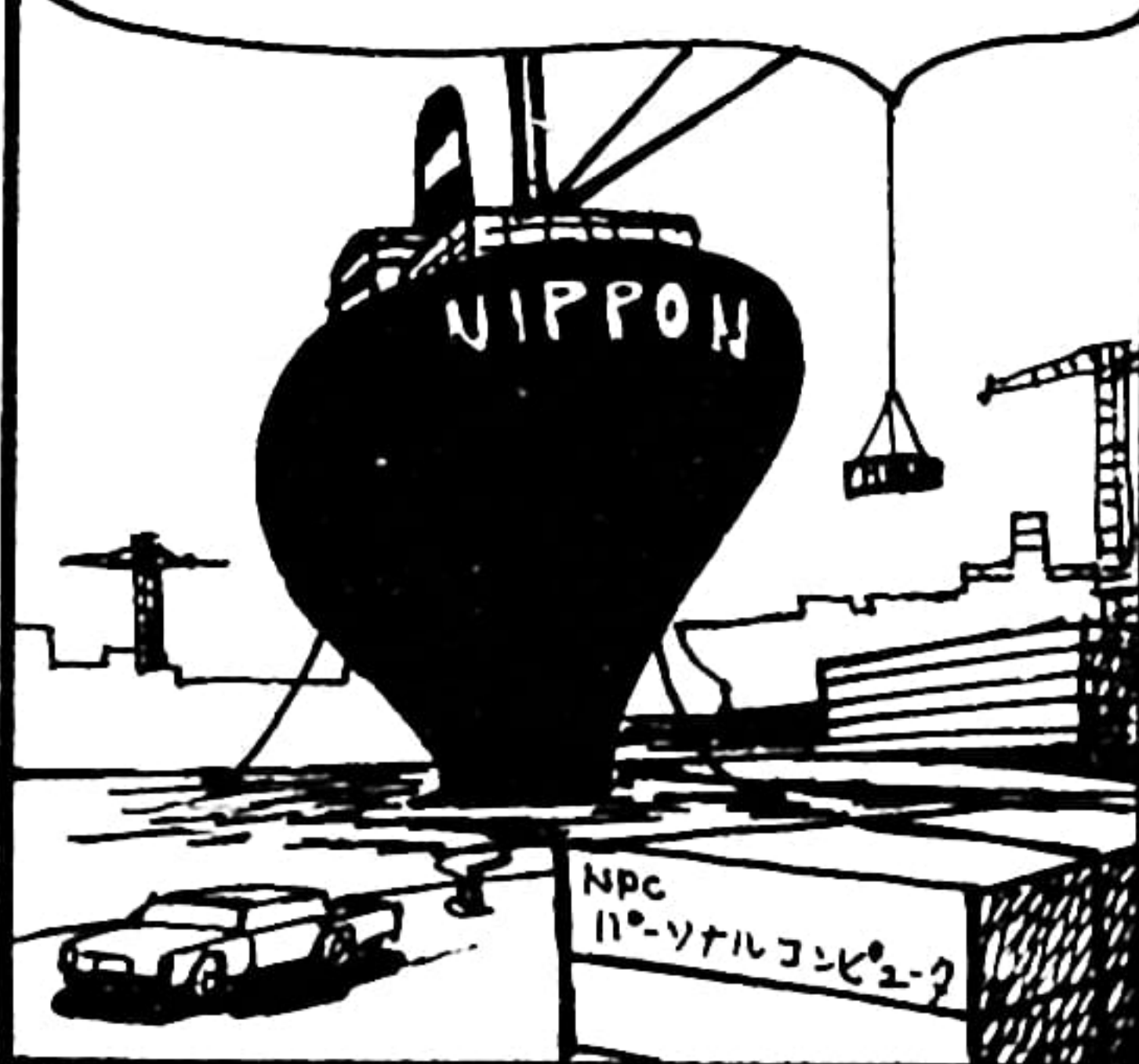
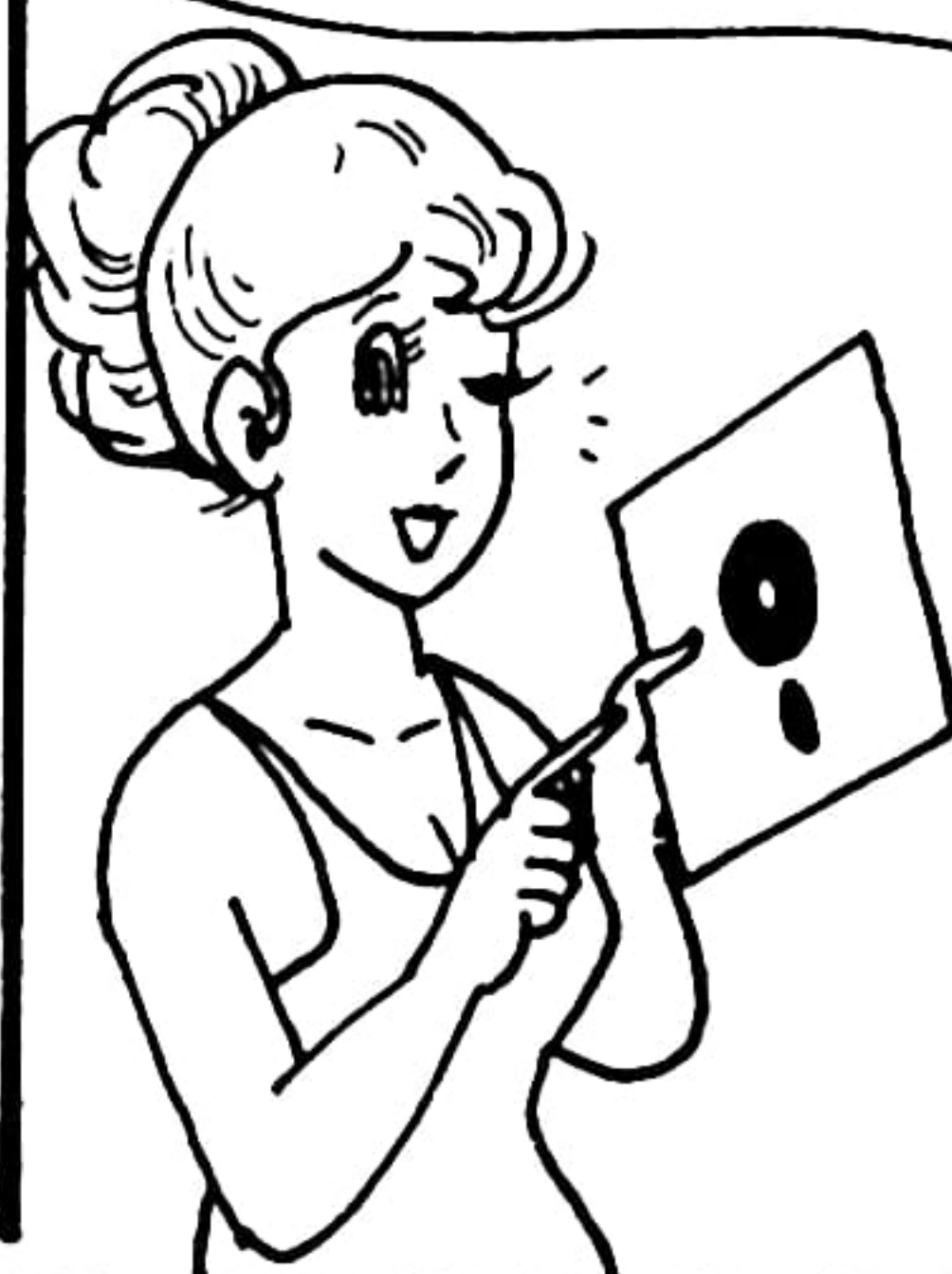
NIPPON

NPC パーソナルコンピュータ

メモリなどの、ICレベルではシリコンバレーのIC業界と日本の業界で多少の摩擦があるようですが、パソコンのレベルではまだそれがありません。

なぜでしょうか？　それは今の日本のパソコンを輸出しても現地ではそう売れないからです。

オー、ニホンノパソコンデスカ？

なぜ売れないかというと、日本のパソコンはROM型で、BASICしか走らないものが多いからです。

オーノー、コレ BASIC オンリー？ワタシ、買イマセーン。

今8ビットではオールRAMの上でCP/M というプログラムが走らないパソコンはパソコンでないというのがあちらではほとんど常識なのです。

CP/Mは0番と1番の2つのトラックの中に入っています。

ここで、CP/M をくわしくお話するわけにもいきませんが、これから使うパソコンのIF 800-30（沖電気）がたまたまこのCP/M を使っているので、お話していく途中でだいたいどんなものかわかってもらえると思います。

このパソコンではBASICはもちろんのこと、このような言語も走るのです。

BASIC、ED、ASM、LOADはパソコンを買った時の標準システムの中に入っています。

O BASIC(OKI BASIC)

- ED(エディタ)
- ASM(8080アセンブラ)
- LOAD(ローダ)

- WMIF(ワードマスター)
- M80(Z80アセンブラ)
- L80(ローダ)

- WMIF(ワードマスター)
- PASCAL/MT+(パスカルコンパイラ)
- MTPLUS 000 〜 MTPLUS 006（オーバーレイ用プログラム）
- PASLIB(パスカルランタイムプログラム)
- LINKMT(リンカー)
- ⋮

（ワードマスターはEDよりも楽にソースファイルをつくれます。）

CP/MのQ&A

＊CP/Mって要するに何ですか

一言でいうとディスクオペレーティングシステム用の"ソフト"です。そして、そのソフトは8080のマシン語で作られています。BASIC言語を使うためにはBASICインタプリンタというマシン語プログラムが必要ですが、そのプログラムを作る時にこのCP/Mを使うと非常に効率的だということです。

(1) CP/Mの"本体"はディスク2トラック分だけで非常にコンパクトである。

(2) PIP（ピップ）コマンドでフロッピーやRS-232Cなどへの入出力が簡単にできる。

(3) システムコールと呼ばれるサブルーチン群があり間接的にI/Oをハンドリングするのでプログラム開発中はI/Oの仕様を気にすることはない。

CP/Mの走るパソコンのシステムディスクにはED（エディタ）、ASM（アセンブラ）、LOAD（ローダ）、DDT（デバッガ）が標準的についていますが、これは本来CP/Mがマシン語開発用のソフトだからです。BASICや簡易言語のプログラムを走らせるだけでは非常にもったいない感じです。

CP/Mが使えれば、そのソフト、ファイルの互換性が保たれるので、急速に普及したOSです。

＊CP/Mのうえを走るものは

拡張子が.COMファイル（本当にマシン語になっているもの、ただファイル名に.COMを勝手につけてもダメ）のものはファイル名↵で即走ります。

BASICインタプリタ、ED、ASM、LOAD、DDT、PIPなどはすべて.COM形式のファイルとしてディスクの中に入っています。

CP/Mで使えるソフトは、ハードの制約を越えて、どのハードでも使える、というのがうたい文句です。また、CP/M上ではBASIC以外の高級言語、たとえばCOBOLやFORTRANなども使えるのも魅力です。

アセンブラやエディタを使うマシン語プログラム開発ツールとしての特色、効用以外の、CP/Mのメリットはなんなのですか。

＊CP/Mのハード環境条件

(1) 2ドライブのディスクがあること。

(2) オールRAM型のパソコンであること。

(3) CRTがあること。

(4) プリンタがあること。

初心者はひとつのシステムとして同じメーカーのものを使った方が、メーカーの情報サービスセンターのアドバイスを受けられて、無難です。

CP/M(Control Program for Micro-computer)

CP/Mマシン相互間でのソフト・ファイルがコンパチブルで使えるという点が一番でしょうね。

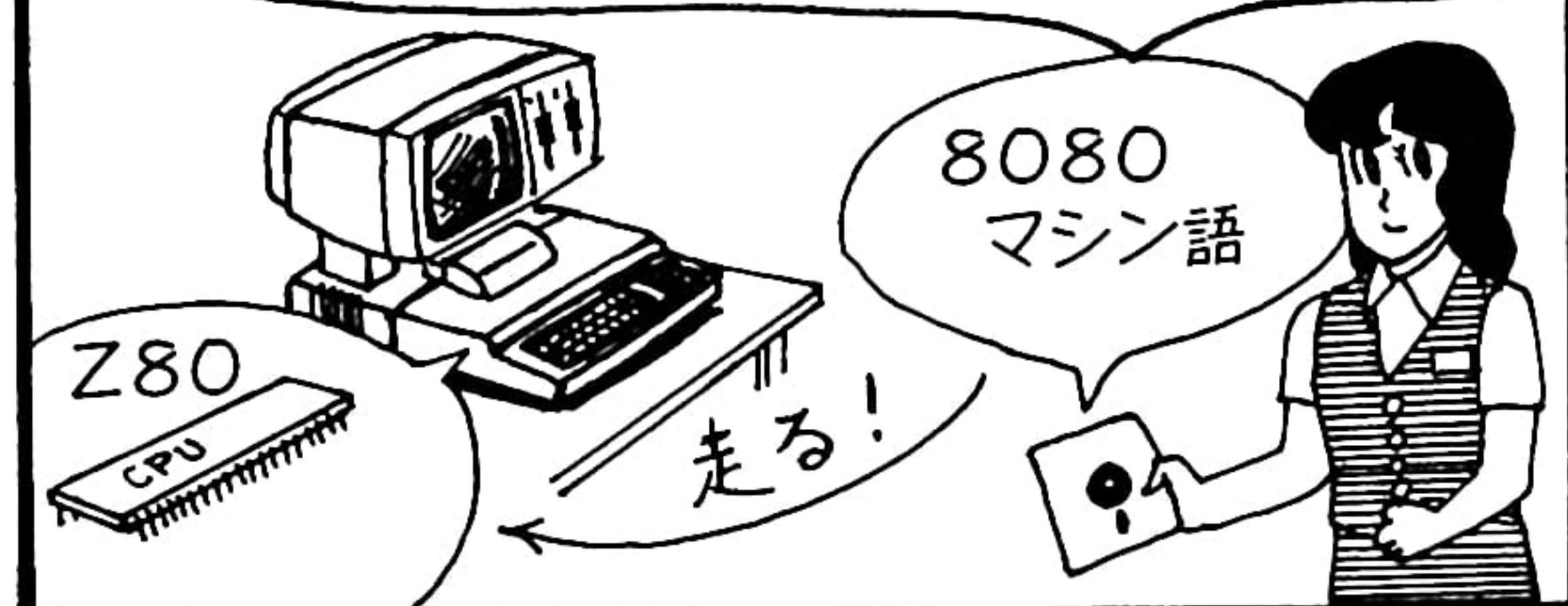
②8080アセンブラを走らせよう

標準システムディスクに入っているのは8080のアセンブラとローダなのです。パソコンのCPUはZ80なので8080のマシン語はそのまま使えるという寸法になっています。

8080マシン語

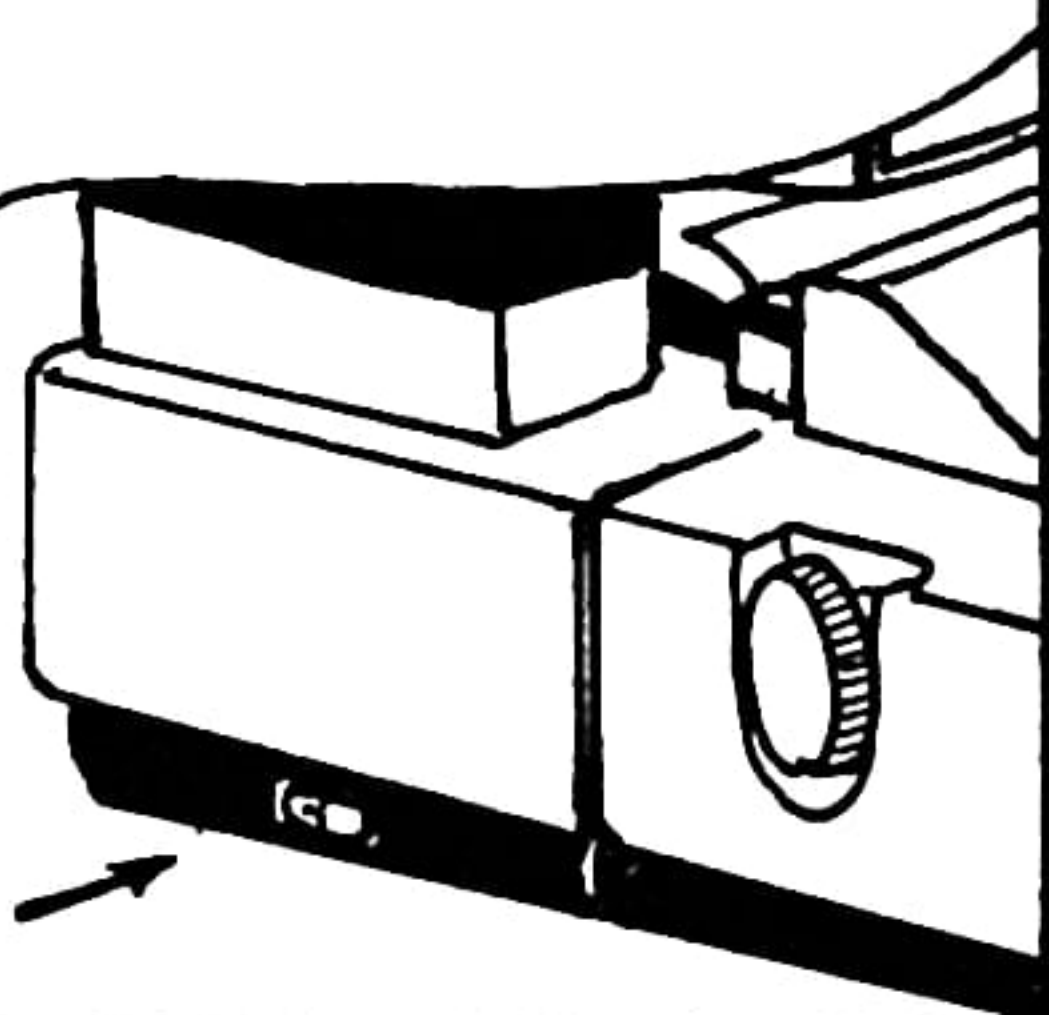
Z80

CPU

走る!

FD

CP/Mを使うにはディスク付オールRAM型パソコンが必要です。

このパソコンはオールインワン方式で電源スイッチを1回入れるだけですむのが便利ですね。

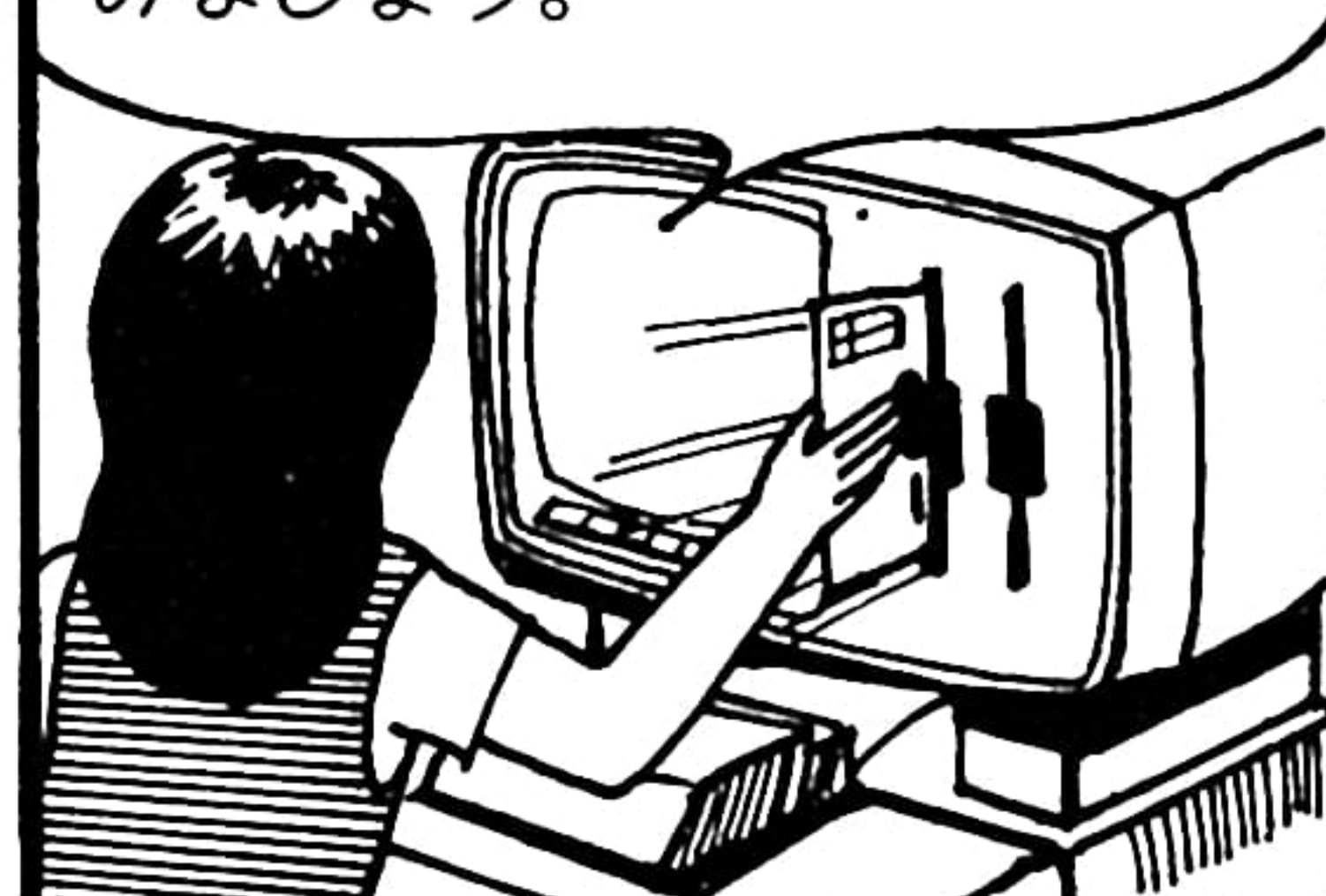
電源スイッチ

まず、CP/Mのシステムを立ち上げます。A>はCP/Mのプロンプト（コマンド待ち状態）でAはドライブ1を示しています。

if800/30 62K-CPM 2.2 REV 1.1
created:16.Apr.82
copyright(c), 1981
Digital research

A>▮

カチャ

カチャ

では、このパソコンでニーモニックプログラムをマシン語に変換するという作業をしてみましょう。

今、しようとしているのはこの.COM形式のファイルをつくることなのです。

ニーモニックでプログラムする
↓
アセンブルする
↓
ローダにかける
↓
.COMファイルを実行する

この.COMは拡張子（エクステンション）でこの場合プログラムがロード後即実行可能なものであることを示しています。.COMファイルは第一番目のOPコードが100Hにロードされます。

100H

TPA
ユーザープログラムエリア

システムディスクにはCP/Mのほかソフト開発ツールとしてのプログラムも入っていなければなりません。

FDDUTY.COM
ED.COM
ASM.COM
LOAD.COM
PIP.COM
⋮

このパソコンを買った時ついているものです。

おニューのFDはFDDUTY（フロッピー・ユーティリティー）で初期化してやらねばなりません。

カチャ

カチャ

A>FDDUTY

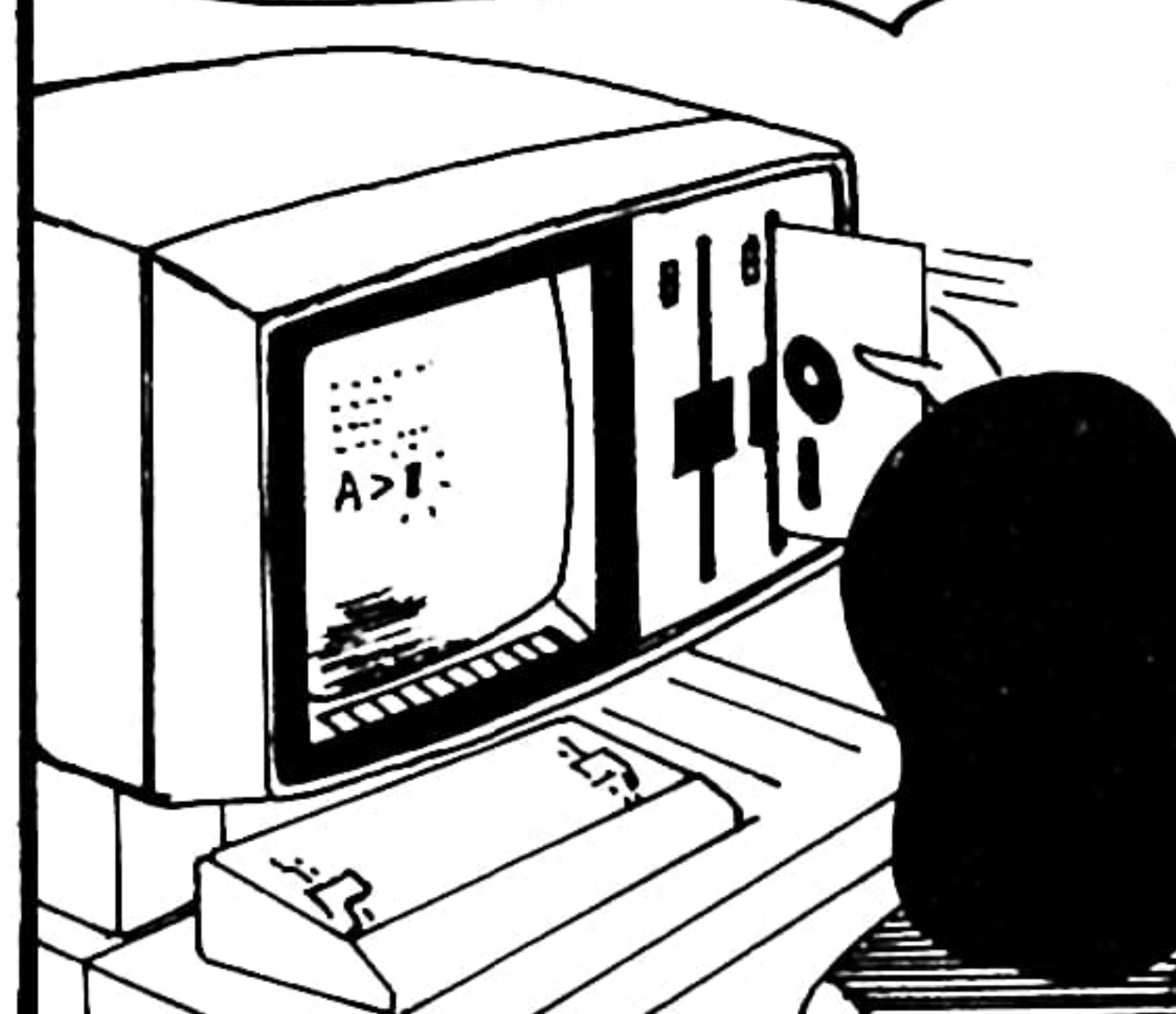
.COMはいりません。

そして RETURN

カチャ

カチャ

ドライブ2に、おニューのFDを挿入します。

A>▮

まず、一番にしておかなくてはならないのがこのシステムディスクのバックアップを作っておくことです。マシン語の場合チョンボによっていつプログラムが暴走してフロッピーがオシャカになるかわからないからです。

システムFD

処女FD

FDDUTYがロードされると、このようなメッセージが表示されます。

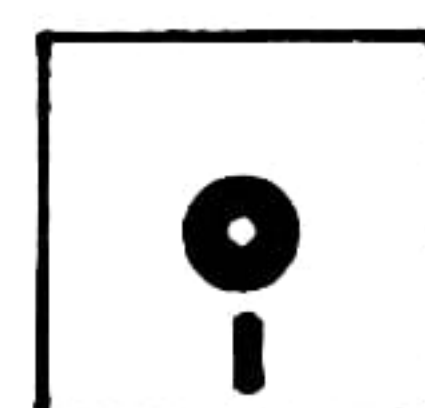
A>FDDUTY
*** if800 model30 cp/M***
*** FDD UTILITY PROGRAM Rev 1.0*
<< program select >>
1. Address format write
2. Volume copy

9. Program end
*** select function

このメニューを見て、1～9までの数字をインプットすればよいわけです。初期化はメニューから1をインプットすればよいことがわかります。

9. Program end
*** select function 1

ここに表示

カチャ 1 RETURN

1を入れると、ドライブBにFDDをセットせよ、と表示されます。セットを確認してRETURNキーを押します。

*** Select function 1
Mount a disk on drive B:

CP/MではB:はFDドライブNo.2のことです。

RETURN

これでOK。トラックNo.が次々に表示されて処理が進みます。

Mount a disk on drive B:
Formatting …type(1S-1D)
Track - 3

ここが0から76まで次々とカウントアップされる。

カチャ カチャ カチャ

この初期化は案外早く終わります。

カチャ カチャ

トラック76までで処理が終わると、再びメニューのセレクトモードになります。次は2を入れてドライブA:に入っているFDの内容をそっくりコピーしてしまいます。

カチャ カチャ

コピーが終わったら、またメニューモードに戻りますので9を入れて作業を終わります。モードはA>です。

では、CP/Mのコマンドを少し紹介します。まずFDの中身を知りたい時のコマンドです。

RETURN のキーインは↲で表現します。

DIR↲でドライブA:に入っているFDのファイル名が表示されます。

カチャ

```
A: MOVCPM  COM : PIP    COM :
A: ED      COM : ASM    COM :
A: STAT    COM : SYSGEN COM:
A: BIOS    ASM : CBIOS  ASM:
A: FDDUTY  COM : KANJI  COM:
A: OBASIC  COM :
       :
```

ログインディスクA>をB>に変更したい時はB:↲です。Bがログインディスクとなります。

A B

A>B:
B>

もう1度A:↲とすると再びAに戻ります。ではCP/Mでソフト開発のマネをしてみたいと思います。テスト用のニーモニックプログラムをCP/Mの上で走らせてみたいと思います。

ニーモニック

6桁のたし算

このニーモニックは8088用でした。CP/Mのアセンブラは8080のニーモニックなのです……。

```
      ORG 2011
      LXI B,2000
      LXI H,2003
      XRA A
      MVI E,03
Loop: LDAX B
      ADC M
      DAA
      STAX B
      DCR E
        :
```

手順としてはまずED（エディタ）を起動します。ED.COMファイルがありましたね。このようにキーインをします。
ED␣TOMOKO.ASM↵

```
A>ED
TOMOKO.ASM
```

これでEDが起動し、メインメモリ内に編集用のバッファができます。ファイル名となるTOMOKO.ASMの.ASMは必ずつけておく必要があります。NEW FILEと＊が表示しました。

カチャ

```
A>ED TOMOKO.ASM

NEW FILE
      :*
```

＊はEDのプロンプトです。OLDファイル名で起動した時は、Aコマンドでファイルからバッファへデータを移した後に編集しますが、NEWファイルの時はそのままＩコマンドでインサートモードに入ります。

```
A>ED TOMOKO.ASM
NEW FILE
      :*
      :i
    1:
```

i↵

インサートモードでは文番号が自動的に生成されます。1:、2:、3:など。

```
    :*
    :i
  1:ORG 100H
  2:LXI B,200H
  3:XRA A
```

カタ カタ

ORG␣100H↵
LXI␣B,200H↵
XR␣A↵

インサートモードの終了はコントロールZです。CTRLキーとZを同時に押します。テキストなどでは この操作を↑Zとか∧Zとか表現しています。

```
14:JMP LOOP
15:DONE:HLT
16:END
17:
   :*
```

ここで∧Zをキーインした。

次のコマンドを受け付けるプロンプトモードになった。

バッファ内にインサートされたニーモニックプログラムをFDDにファイルするコマンドはEです。

カチャ カチャ

```
16:END
17:
   :*E
```

E↵

コマンドEが実行されるとFDにはED起動時につけたTOMOKO.ASMとTOMOKO.BAKという2つのファイルが生成されます。

1 2

カチャ カチャ

.ASMファイルには今バッファにインサートした新しい内容が入り、.BAKファイルには編集前の内容がバックアップとして残されます。

```
16:END
17:
   :*E
A>DIR␣TOMOKO.*
A: TOMOKO BAK:
       TOMOKO ASM:
A>
```

DIR␣TOMOKO.*↵

このTOMOKO.ASMをアセンブルするためにASM（アセンブラ）を起動します。ED起動時には.ASMをつける必要がありましたが、今度は.ASMをつけるとエラーです。

カチャ

```
A>ASM TOMOKO
```

ASM␣TOMOKO↵

カタ カタ

FDからASM.COMとTOMOKO.ASMが呼び出されてアセンブルされます。FDDにTOMOKO.PRNとTOMOKO.HEXの2つのファイルを作成してアセンブルが終わります。

```
A>ASM TOMOKO
CP/M ASSEMBLER-VER 2.0
0117
000H USE FACTOR
END OF ASSEMBLY
A>
```

カチャ カチャ

DIR␣TOMOKO.*↵とキーインするとTOMOKOに関するファイル名だけが表示されます。

```
A: TOMOKO ASM:
   TOMOKO BAK:
   TOMOKO PRN:
   TOMOKO HEX:
A>
```

TOMOKOで4つのファイルができました。＊はこの場合すべての拡張子という意味です。

次にLOAD（ローダ）を使って、TOMOKO.COMファイルを作ります。ローダはFDからTOMOKO.HEXファイルを呼び出し.COMファイルをFDに生成します。LOAD␣TOMOKO.HEXと.HEXをつけるとエラーになります。

```
A>LOAD TOMOKO
```

LOAD␣TOMOKO↵

カタ カタ

このTOMOKO.COMはCP/M上で即実行できるプログラムです。

A>TOMOKO

TOMOKO↵

TOMOKO.COMファイルができているはずです。DIRでみてみましょう。

A> DIR TOMOKO.COM
A: TOMOKO COM
A>

DIR␣TOMOKO.COM↵

このように表示されてLOADプログラムの処理が完了します。

A>LOAD TOMOKO

FIRST ADDRESS 0100
LAST ADDRESS 0116
BYTES READ 0017
RECORDS WRITTEN 01

A>

印字はCP/MではPIP(ピップ)コマンドを使います。PIP␣LPT:=までが印字のためのコマンドで、その後に印字したいファイル名をキーインすればよいのです。

A>PIP LPT:=TOMOKO.ASM

PIP␣LPT:=TOMOKO.ASM↵

そこでTOMOKO.ASMを再編集して結果の見えるものにするつもりです。まず今作った各TOMOKOファイルを印字することにします。

でもあれですね。このプログラムは3バイトのたし算なのですが、まだデータが入っていないし、結果を表示するルーチンもないので、このままでは意味がありません。

A>TOMOKO
A>

```
ORG 100H
LXI B,2000H
LXI H,2003H
XRA A
MVI E,03H
LOOP:LDAX B
ADC M
DAA
STAX B
DCR E
JZ DONE
INX B
INX H
JMP LOOP
DONE:HLT
END
```

これがTOMOKO.ASMの印字内容です。インサートモードの時に表示されていた行番号は、ありません。

数年前まではこのソースプログラムを紙テープに落としていたのです。

これでTOMOKO.ASMのファイルがログインディスクA:のFDから読み出されて、印字されます。

ビィーッ
ビィーッ

```
0100              ORG 100H
0100 010020       LXI B,2000H
0103 210320       LXI H,2003H
0106 AF           XRA A
0107 1E03         MVI E,03H
0109 0A           LOOP:LDAX B
010A 8E           ADC M
010B 27           DAA
010C 02           STAX B
010D 1D           DCR E
010E CA1601       JZ DONE
0111 03           INX B
0112 23           INX H
0113 C30901       JMP LOOP
0116 76           DONE:HLT
0117              END
```

これはハンドアセンブルリストと同じです。テキストでよく見かけるこの書式はPRNファイルだったのです。

同様にして PIP␣LPT:=TOMOKO.PRN↵

カチャ
カチャ

このプログラムは6桁（3バイト）のたし算でした。

```
  34 28 57
+ 18 39 18
----------
  52 67 75
```

そこでまず2000H～2005Hの6バイトにデータを入れるルーチンが必要になります。

2000H 57 28 34 18 39 18

データ

まず定数をメモリに書き込むルーチンです。(データのロード)

```
ORG 100H
LXI H,2000H
MVI M,57H
INX H
MVI M,28H
INX H
MVI M,34H
INX H
MVI M,18H
INX H
MVI M,39H
INX H
MVI M,18H
```

あまりスマートではありませんが、これでもいいはずです。

インテル8080の命令表を参照してください。

この数値は10進数のつもりですがMVI M,57Dなどと書くとアセンブルされた時39Hに変換されてしまいます。そこでMVI ,57Hと書いています。

39 ← アセンブラ ← 57D
39 ← ← 57
57 ← ← 57H

次にTOMOKOの部分を実行した後の2000H～2002Hの内容を表示するルーチンをつけ加えます。

34 28 57
+18 39 18
52 67 75

表示 75 67 52 (2000H) 18 39 18 (2003H)

表示はCP/Mのシステムコールと呼ばれるサブルーチンを使います。１バイトのアスキーコードをキャラクタで表示します。

```
MVI C,2
MOV E,A
CALL 5
```

これでAの内容が表示されました。

そこで5・2・6・7・7・5とそれぞれのキャラクタを個別にアスキーコードで扱う必要があります。この場合は全部数字ですから頭に3をつけて35、32、36、37、37、35とすればアスキーコードになります。ブランク(スペース)コードは20です。

52 → 05 → 35
52 → 02 → 32

被加算数は2000Hから3バイト加算数は2003Hから3バイトにデータとして入れておき答は2000Hから3バイトに入ります。プログラムが走ったあとは被加算数の代わりに答が入っています。

2000H 答 ⇐ 実行 2000H 被加算 2003H 加算

この2つのルーチンを先に作ったTOMOKO.ASMにつけ加えてTAKAKO.ASMというソースファイルを作ってみます。

エディタを使う：データのロード ＋ TOMOKO・ASM ＋ 答の表示

答は52、67、75の3バイトですが52をAレジスタに入れてもダメです。52はアスキーコードではRです。つまりこの場合R・g・uと表示されてしまうのです。

52 67 75 → アスキーコードとみなして表示したら… → R g u

?!

1バイトで表現される52を5と2に分解してアスキーコードの35と32にします。アスキーでファイルに書き込む時もこのように多くのバイトが必要になると思われます。

内部表現

52	67	75

↓ アスキー表現

35	32	36	37	37	35

具体的にはAレジスタを使ってビット操作します。まず、答の先頭の2002Hの内容(52)をAレジスタに入れます。

A
52
2002H 52
プログラムはあとで考えます。

次にAレジスタの下の4ビットをマスクするためにF0とのANDをとります。

52
A
0101 0010
F0
1111 0000
ANDをとる
結果は、Aに
0101 0000
50
0になった！

この状態でAレジスタの内容は52から50に変わっています。ここで、さらにAレジスタを4ビット右に回転してやると05になります。

0101 0000 RRC（回転）
0010 1000 RRC
0001 0100 RRC
0000 1010 RRC
0000 0101

この05と30とのOR(オア)を取ってやればAレジスタの内容は35となります。これでキャラクタ'5'のアスキーコード35ができました。

A
05
0000 0101
30
0011 0000
ORをとると
Aレジスタは
0011 0101
35 となる

このAレジスタの内容35Hを使ってシステムコール（ファンクション2）をすれば'5'が表示されるわけです。

35 表示用コード

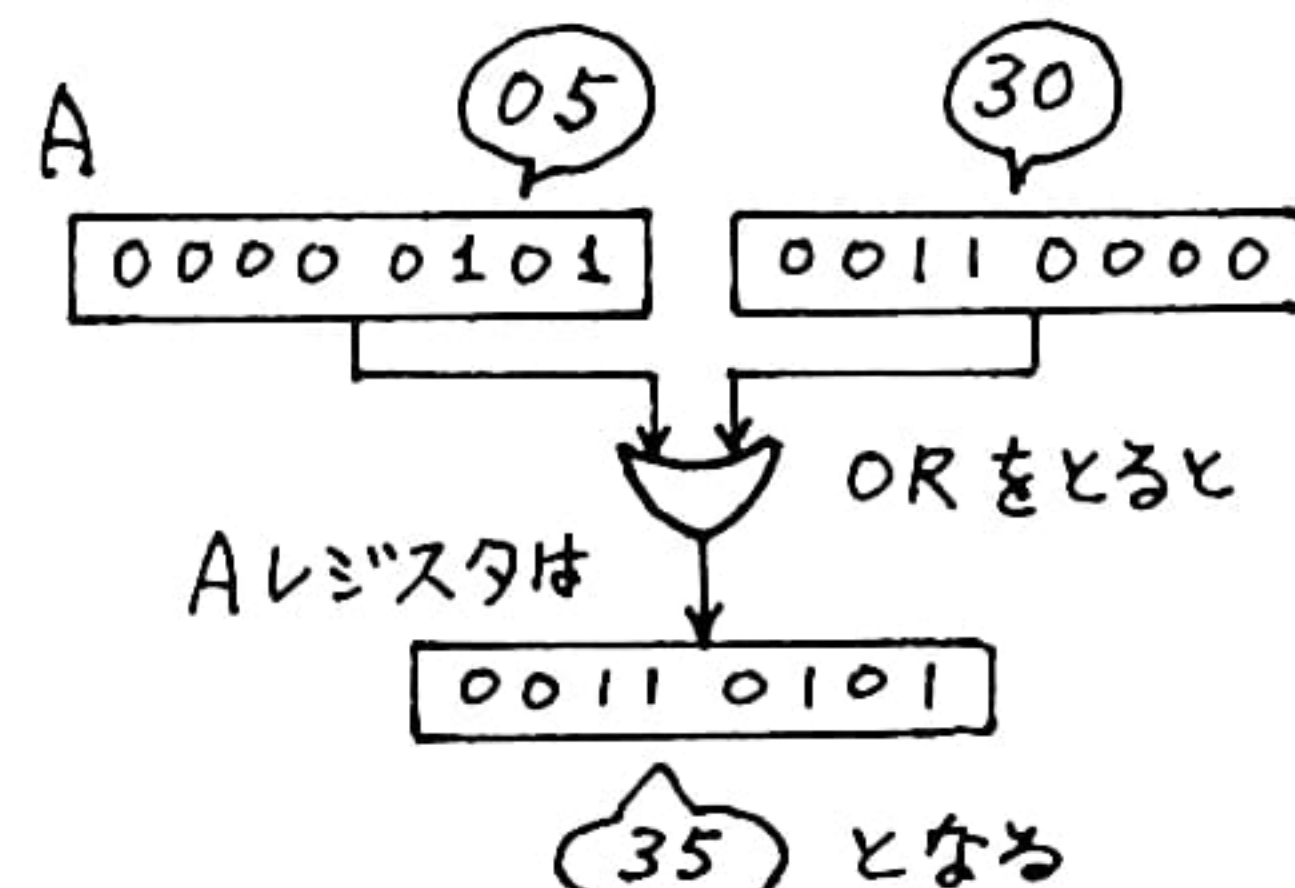
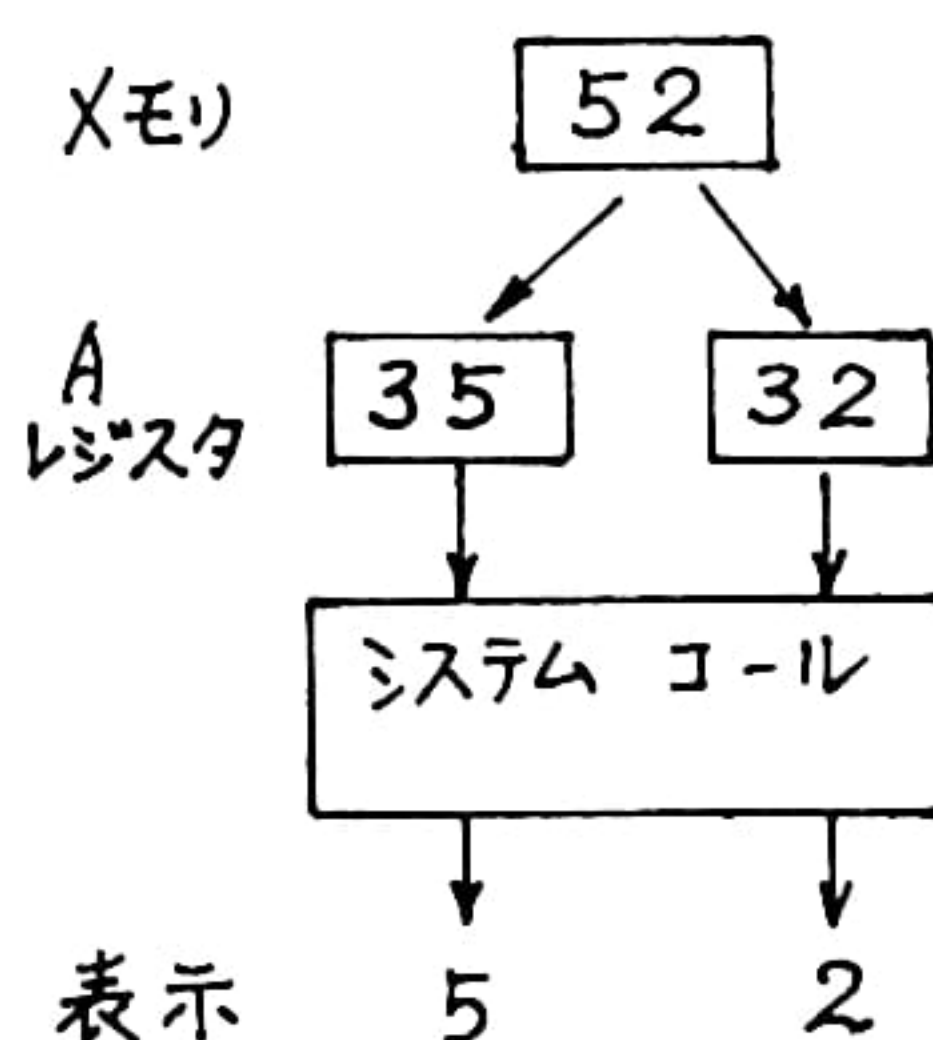
```
MVI C,2
MOV E,A
CALL 5
```

35
このシステムコールはCP/Mに用意されたサブルーチンです。

下桁の2はもう少し簡単です。もう一度2002Hから52をAレジスタに入れます。

A
52
2002H

今度は上4ビットを0FとのANDを取ってマスクし、これと30とのORを取ってやれば、'2'のアスキーコード32が得られます。

A
0101 0010
52
0000 1111
0F
AND
A
0000 0010
02
0011 0000
30
OR
A
0011 0010
32

32Hとなった Aレジスタの内容をEレジスタに入れ、システムコールすれば'2'が表示されます。

```
MVI C,2
MOV E,A
CALL 5
```

32

これで1バイト分の表示ができました。

メモリ 52
Aレジスタ 35 32
システム コール
表示 5 2

区切りのためにブランクを表示します。ブランクのためのアスキーコードは20Hです。

```
MVI C,2
MOV E,20H
CALL 5
```

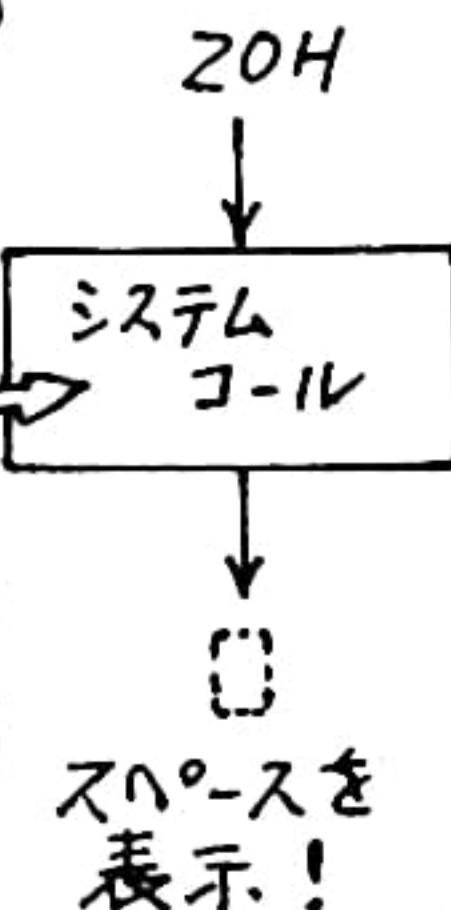
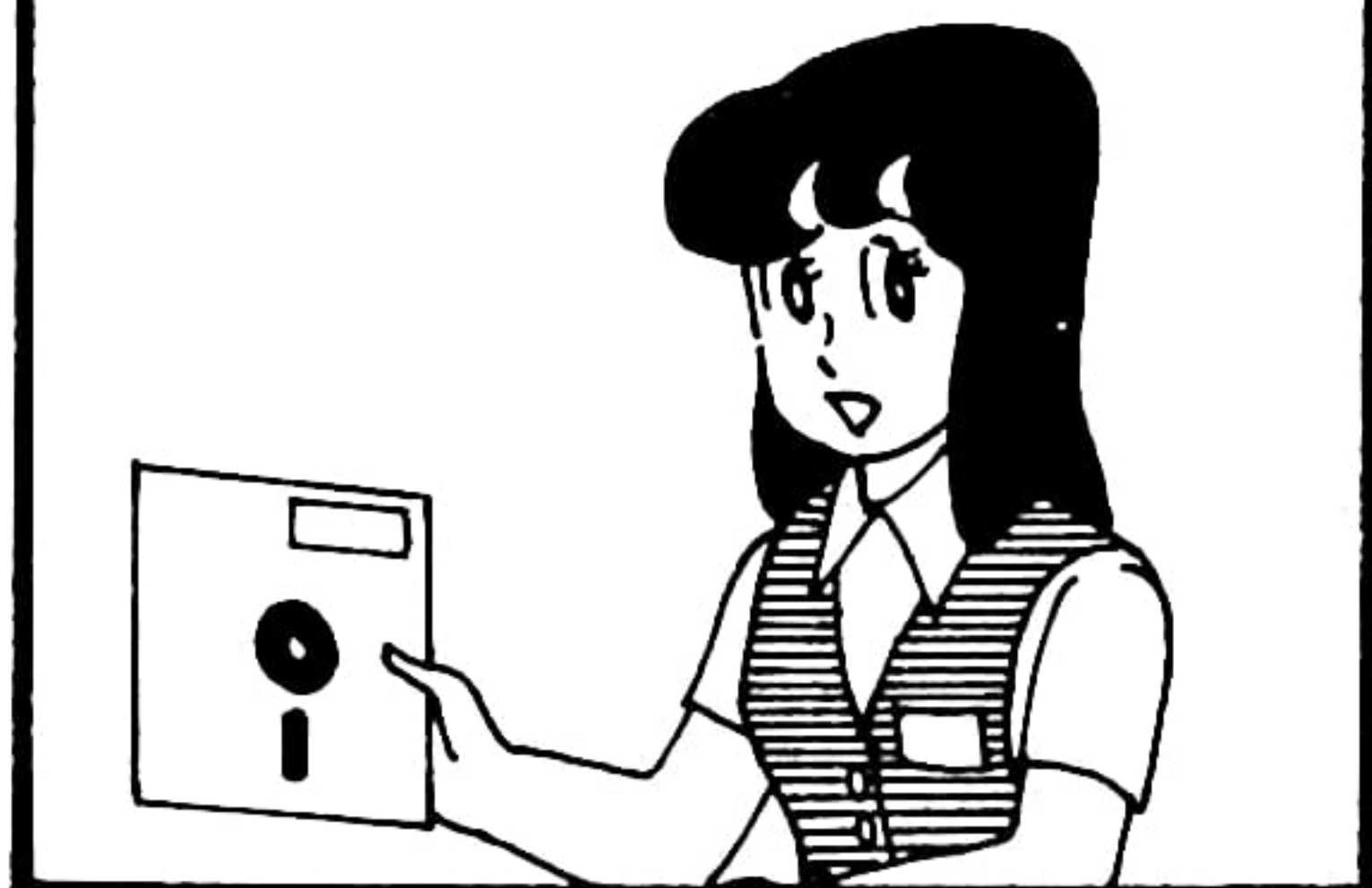
20H
システム コール
スペースを表示！
スペースもキャラクタなのです。

あと2バイト分のデータもこれと同じことをやればよいわけですからこの部分をサブルーチンにします。

2002Hをセット
CALL SATOMI
2001Hをセット
CALL SATOMI
2000Hをセット
CALL SATOMI

SATOMI
上4ビット表示
下4ビット表示
スペース表示
RET

表示部のルーチンができました。アドレス指定にBCレジスタペアを使いシステムコールでCレジスタを使うため、PUSH、POP命令でデータを守っています。

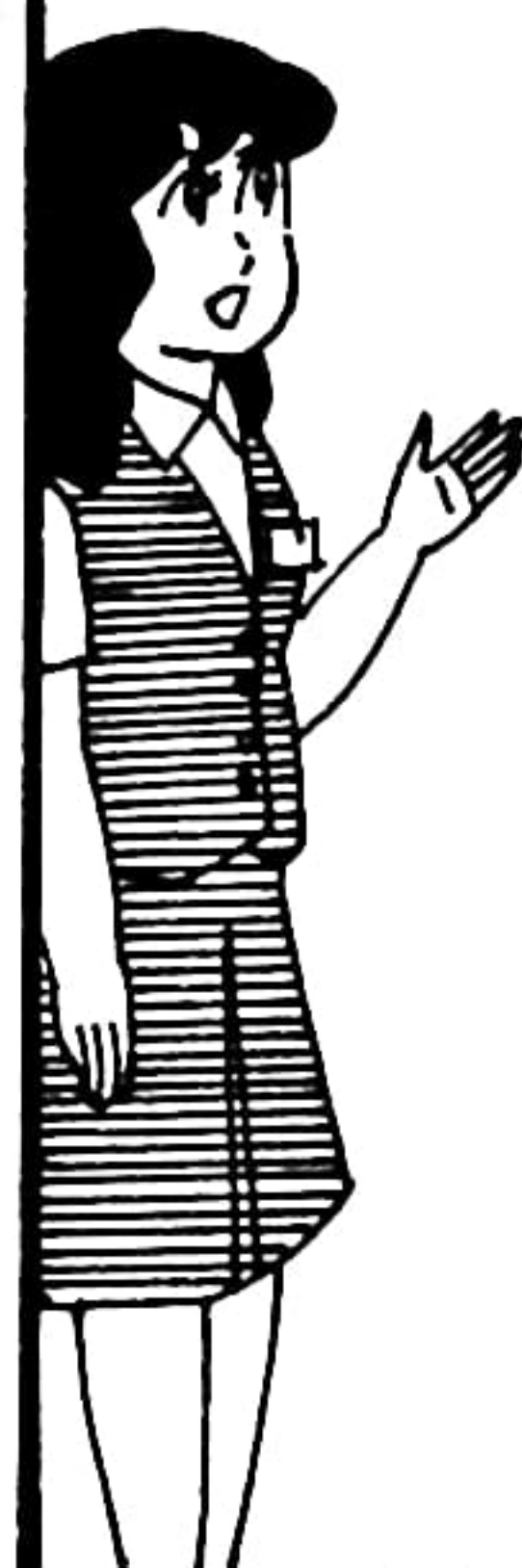
B C
PUSH B
スタックエリア
C
B
POP B
B C

ではED(エディタ)を使ってTOMOKO.ASMにこれらをくっつけてみます。

ED␣TOMOKO.ASM↵

```
A>ED TOMOKO.ASM
     :*
```

今度はNEW FILEとは表示されません。

サブルーチンです。

ゼロが必要

```
SATOMI:ANI␣0F0H
       RRC
       RRC
       RRC
       RRC
       ANI␣0FH
       ORI␣30H
       PUSH␣B
       MVI␣C,2
       MOV␣E,A
       CALL␣5
       POP␣B
       LDAX␣B
       ANI␣0FH
       ORI␣30H
       PUSH␣B
       MVI␣C,2
       MOV␣E,A
       CALL␣5
       MVI␣C,2
       MVI␣E,20H
       CALL␣5
       POP␣B
       RET
```

表示部のメインルーチンです。

```
DONE: HLT
        ↓
      LXI␣B,2002H
LDAX␣B
CALL␣SATOMI
DCX␣B
LDAX␣B
CALL␣SATOMI
DCX␣B
LDAX␣B
CALL␣SATOMI
RET
END
```

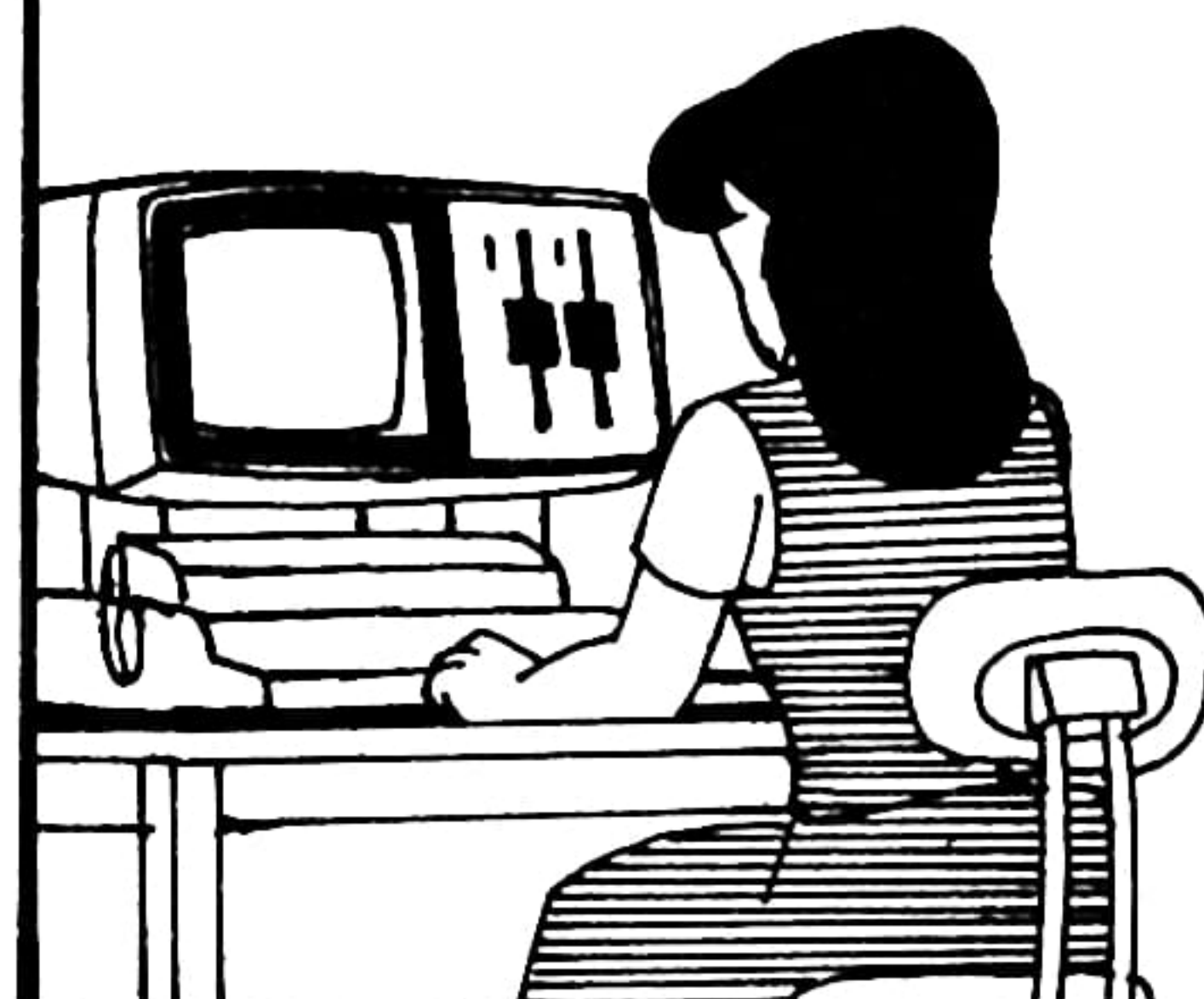
CP/Mへ戻るためのリターン文です。

0(ゼロ)A↵

このAコマンドは、ASMファイルからバッファにデータを移すためのものです。

```
A>ED␣TOMOKO.ASM
      :*0A
      1:
```

これでバッファの中にTOMOKO.ASMの内容が移ります。↵で順次1行ずつ見ることができます。

```
A>ED␣TOMOKO.ASM
      :*0A
     1:                 ↵
     2: LXI␣B,2000H
     2:*                ↵
     3: LXI␣H,2003H
     3:*                ↵
     4: XRA␣A
     4:*
```

数字±nを入れて↵すると±n番目の内容が表示されます。

```
4: XRA␣A
4:*-3          -3↵
1:ORG␣100H
1:*            ↵
2:LXI␣B,2000H
2:*
```

今ORG␣100HとLXI␣B,2000Hの間にデータロードのルーチンを挿入します。インサートモードにするためのコマンドはIです。

```
2: LXI␣B,2000H
2:*I            I↵
2:
```

この場合、リストでみるとLXI␣B,2000Hの上に挿入されていきます。

(モニタ表示)

```
2:  LXI␣B,2000H
2:*I
2:  LXI␣H,2000H
3:  MVI␣M,57H
4:  INX␣H
5:  MVI␣M,28H
 :
13:  MVI␣M,18H
```

ORG␣100Hの次から入れていくことになります。

LXI␣H,2000H↵
MVI␣M,57H↵
INX␣H↵
MVI␣M,28H↵
⋮
MVI␣M,18H

ひとつ戻してDONE␣HLTを表示させせます。

```
13:  MVI␣M,18H
14:
  :*-B
28:
28:*-1
27:`DONE␣HLT
27:*
```

−1↵

±Bコマンドは行の表示をテキストの先頭か末尾に一気にとばすものです。

```
13:  MVI␣M,18H
14:
  :*-B
28:
28:*
```

−B↵

インサートモードの終了は∧Zです。CTRLキーとZキーを同時に押すことを意味します。

```
13:  MVI␣M,18H
14:
  :*
```

∧Z

28番からインサートするので、ひとつ空振りします。↵

```
27: DONE : LXI␣B,2002H
27:*
28:
28:*I
28:
```

↵

I↵

28番には何も入っていません。

確かめます。−1↵

```
27:*SHLT^Z 2002H
28:
28:*-1
27:  DONE : LXI␣B,2002H
27:*
```

DONE␣HLTのHLTをLXI␣B,2002Hに置き換えます。Sコマンドをこのように使います。

```
27:  DONE␣HLT
27:*SHLT ^Z LXI␣B,2002H
28:
28:*
```

SHLT^Z KXI␣B, 2002H↵

カチャ カチャ

∧Zでインサートモード終了します。

```
61:  RET
62:
  :*
```

∧Z

（モニタ表示）

以下前ページのプログラムの順に入れていきます。

```
28:*I
28:  LDAX␣B                 LDAX␣B↵
29:  CALL␣SATOMI            CALL␣SATOMI↵
 :                           :
37:  END                    END↵
38:  SATOMI:ANI␣0F0H        SATOMI : ANI␣0F0H↵
39:  RRC                    RRC↵
 :                           :
61:  RET                    RET↵
62:
```

FDD上のTOMOKO.ASMには今編集したテキストが入り、TOMOKO.BAKには編集前のテキストがバックアップ用として入るのです。

```
61: RET
62:
  :*E
A>▮
```

Eコマンドで今編集したテキストをFDに書き込みます。

```
61: RET
62:
  :*E
```

E↵

カチャ カチャ

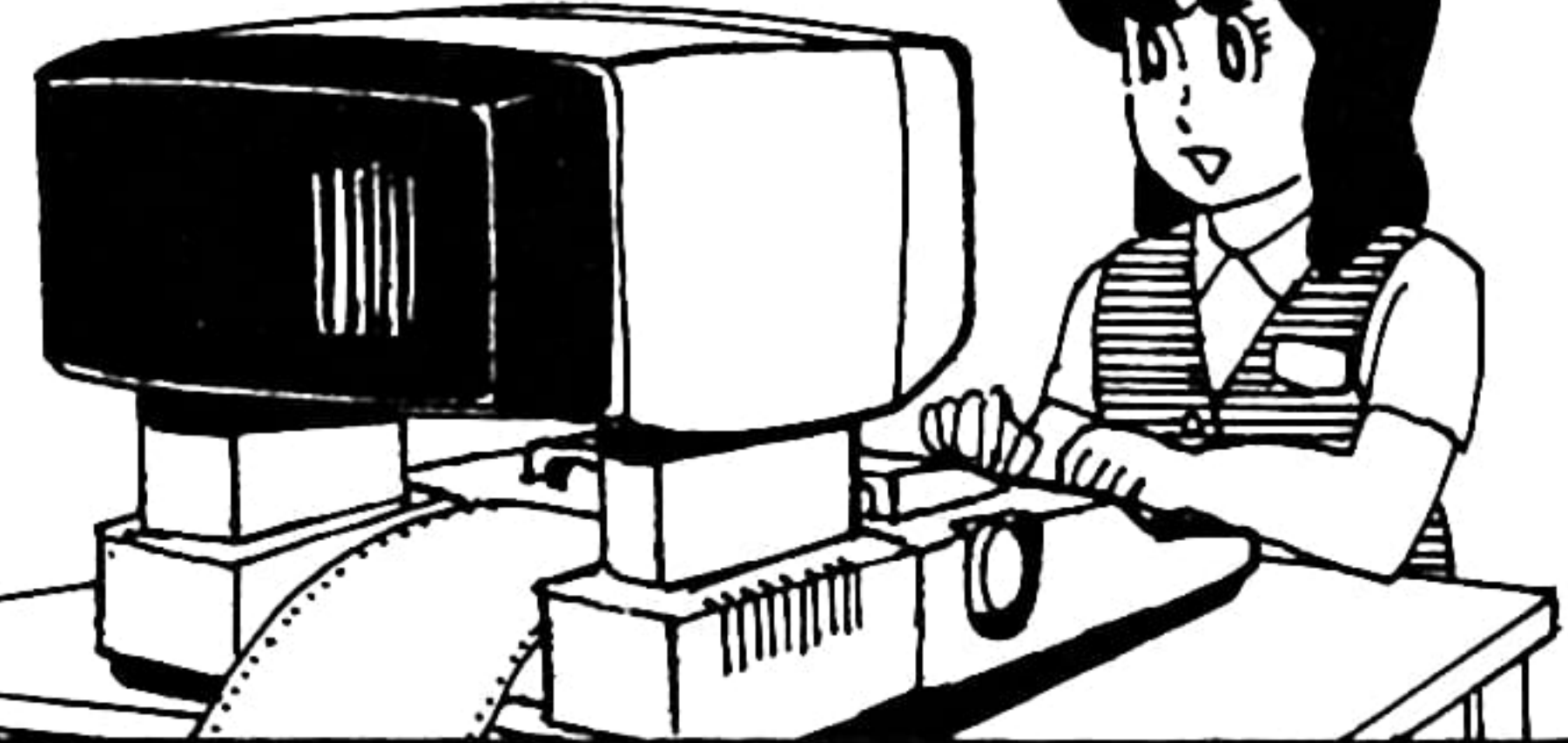
今は使いませんでしたが行を削除したい場合もありますね。この時はKコマンド（キル）を使います。

K↵など

TAKAKO.PRNのプリントです。

A>PIP␣LPT:=TAKAKO.PRN↵

```
0100                ORG 100H
0100 210020         LXI H,2000H
0103 3657           MVI M,57H
0105 23             INX H
0106 3628           MVI M,28H
0108 23             INX H
0109 3634           MVI M,34H
010B 23             INX H
010C 3618           MVI M,18H
010E 23             INX H
010F 3639           MVI M,39H
0111 23             INX H
0112 3618           MVI M,18H
0114 010020         LXI B,2000H
0117 210320         LXI H,2003H
011A AF             XRA A
011B 1E03           MVI E,03H
011D 0A             LOOP:LDAX B
011E 8E             ADC M
011F 27             DAA
0120 02             STAX B
0121 1D             DCR E
0122 CA2A01         JZ DONE
0125 03             INX B
0126 23             INX H
0127 C31D01         JMP LOOP
012A 010220         DONE:LXI B,2002H
012D 0A             LDAX B
012E CD3C01         CALL SATOMI
0131 0B             DCX B
0132 0A             LDAX B
0133 CD3C01         CALL SATOMI
0136 0B             DCX B
0137 0A             LDAX B
0138 CD3C01         CALL SATOMI
013B C9             RET
013C E6F0           SATOMI:ANI 0F0H
013E 0F             RRC
013F 0F             RRC
0140 0F             RRC
0141 0F             RRC
0142 E60F           ANI 0FH
0144 F630           ORI 30H
0146 C5             PUSH B
0147 0E02           MVI C,2
0149 5F             MOV E,A
014A CD0500         CALL 5
014D C1             POP B
014E 0A             LDAX B
014F E60F           ANI 0FH
0151 F630           ORI 30H
0153 C5             PUSH B
0154 0E02           MVI C,2
0156 5F             MOV E,A
0157 CD0500         CALL 5
015A 0E02           MVI C,2
015C 1E20           MVI E,20H
015E CD0500         CALL 5
0161 C1             POP B
0162 C9             RET
                    END
```

このTOMOKO.ASMをアセンブルする必要がありますが、すでにTOMOKOでは.PRN、.HEX、.COMなどのファイルがあるため、まずこれらのファイルを消してしまうかTOMOKO.ASMを別の名前にしてしまわねばなりません。

ここでは別の名前、つまりリネームすることにします。リネーム名はなんでもいいのですが、TAKAKO.ASMとし次のようにキー操作します。REN␣TAKAKO.ASM=TOMOKO.ASM↵

```
A>REN␣TAKAKO.ASM=TOMOKO.ASM
A>▮
```

カシャ

DIRでTAKAKO.ASMを確かめます。DIR␣TAKAKO.ASM↵

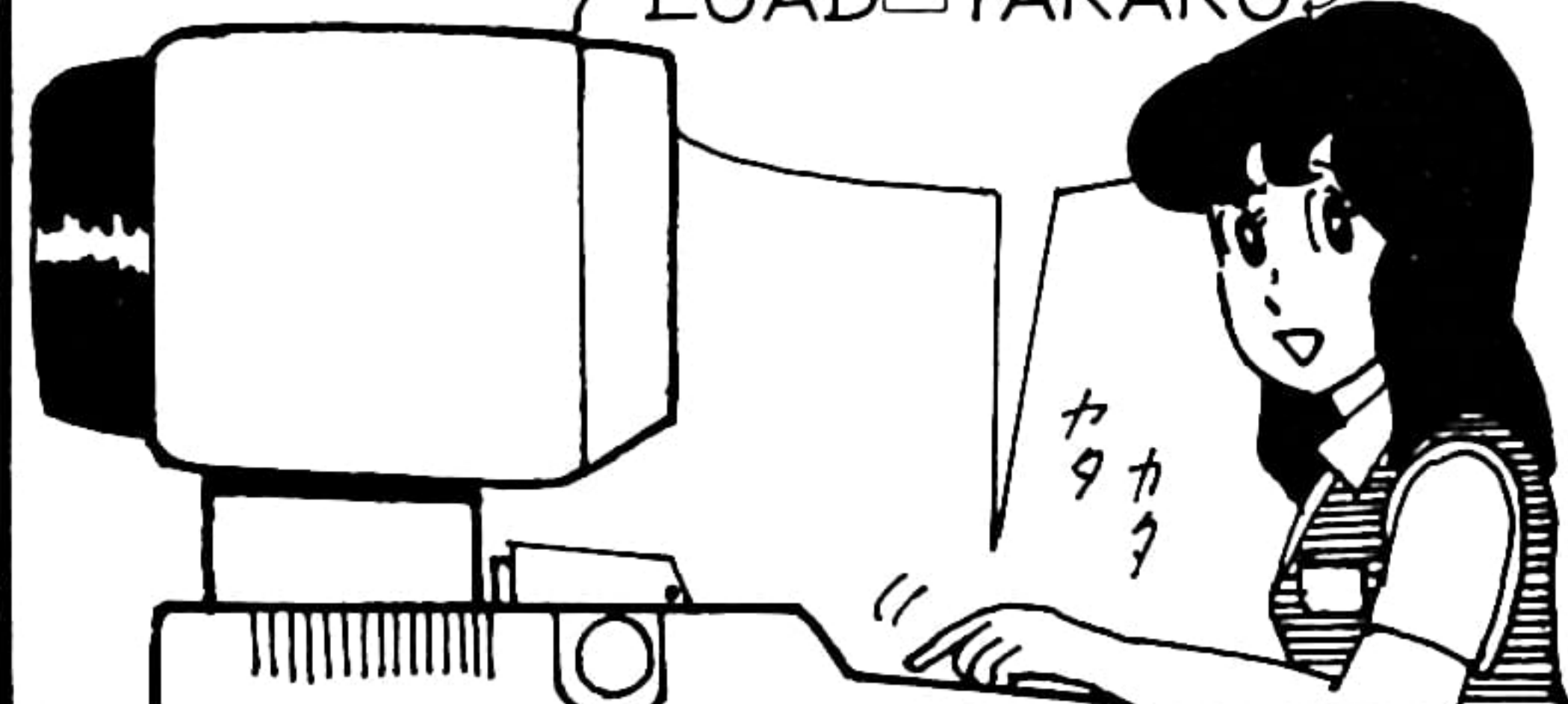
```
A>REN␣TAKAKO.ASM=TOMOKO.ASM
A>DIR␣TAKAKO.ASM
A:TAKAKO.ASM
A>▮
```

TAKAKO.ASMはTOMOKO.ASMとまったく同じ内容のソースファイルです。

このTAKAKO.ASMでアセンブルし、さらに.COMファイルを作ります。

ASM␣TAKAKO↵
LOAD␣TAKAKO↵

カタカタ

ファイルを確認します。DIR␣TAKAKO.*↲

A>DIR␣TAKAKO.*
A:TAKAKO ASM:TAKAKO BAK
:TAKAKO PRN:TAKAKO HEX
:TAKAKO COM
A>

TAKAKO↲とすればCP/MがTAKAKO.COMファイルをFDDから読み出しメモリにロードします。ロード完了後プログラムが走ります。

ではTAKAKO↲

カチャ

カチャ

A>TAKAKO
52 67 75
A>

たし算実行結果の答えです。

あはっ、やった。マシン語のプログラムが走った。

52␣67␣75

誰が何と言おうと私のプログラムもCP/M のうえを走ったのです。

でも、このカンタンなプログラムでもちゃんと表示するまでいくつかのデバッグが必要でした。

ニーモニックプログラムをニラミながらのデバッグ作業は大変です。このためCP/MではDDTというデバッガがあります。

DDTが扱うファイル形式はHEXです。DDT␣TAKAKO.HEX↲とキーをたたきます。

A>DDT␣TAKAKO.HEX
DDT␣VERS 2.2
NEXT PC
0163 0000

カチャ

DDTのプロンプトです。

このDDTにもいくつかのコマンドがあります。まずアドレス2000から200Fまで0(ゼロ)にします。F(フル)コマンドを使います。F2000, 200F,0↲

NEXT PC
0163 0000
-F2000,200F,0

答の入るところをクリアするため。

メモリの内容を見たい時にはD(ダンプ)コマンドがあります。D2000,200F↲

-F2000,200F,0
-D2000,200F
2000 00 00 00 00 00 00 0

先頭アドレス

2000番地のメモリの内容

TAKAKO.PRN(前ページ)を見てください。アドレス100から114までプログラムを走らせます。G(ゴー)コマンドを使います。G100,114↲

2000 00 00 00 00 00 00 0
-G100,114
*0114

この結果2000〜2005にたし算の問題の数値57、28、34、18、39、18が入っていればこの部分のプログラムはOKということになります。D2000,200F↲

-G100,114
*0114
-D2000,200F
2000 57 28 34 18 39 18 00

OKです。

続いて、D2000,200F↲ カチャ、カチャ

では次に012Aまで走らせてどう変化したか見ましょう。G100,12A↲

プログラムエリア

100

12A

2000

データエリア

プログラム処理

75 67 52

うーん、ホントによくできてるわねえ。マシン語のユーツウがかなり楽になりました。

さらにAレジスタやフラグ、B、C、Hのレジスタペア、スタックポインタSP、プログラムカウンタPCの状態を見ることができるのです。Xコマンドです。X↲

A	F
B	C
D	E
H	L
SP	
PC	

このあたりROM型パソコンのMONモードで使ったコマンドに似ていますね。

F…フル

D…ダンプ

G…ゴー

X…イグザミン

2000～200F番地の16個のデータが表示されました。先頭の2000番地とそのデータ75が見えます。75の次の67は2001番地のデータです。

```
-G100,12A
*012A
-D2000,200F
2000 75 67 52 18 39 18 00 00
```

この部分が変わっています。前ページとくらべてください。

なおDコマンドでは1行16個ずつのデータが表示されますが、右端の変な表示はデータをキャラクタとして示したものです。たとえば57はW、28は）、34は4と表示されています。キャラクタ以外のデータは・と表示します。

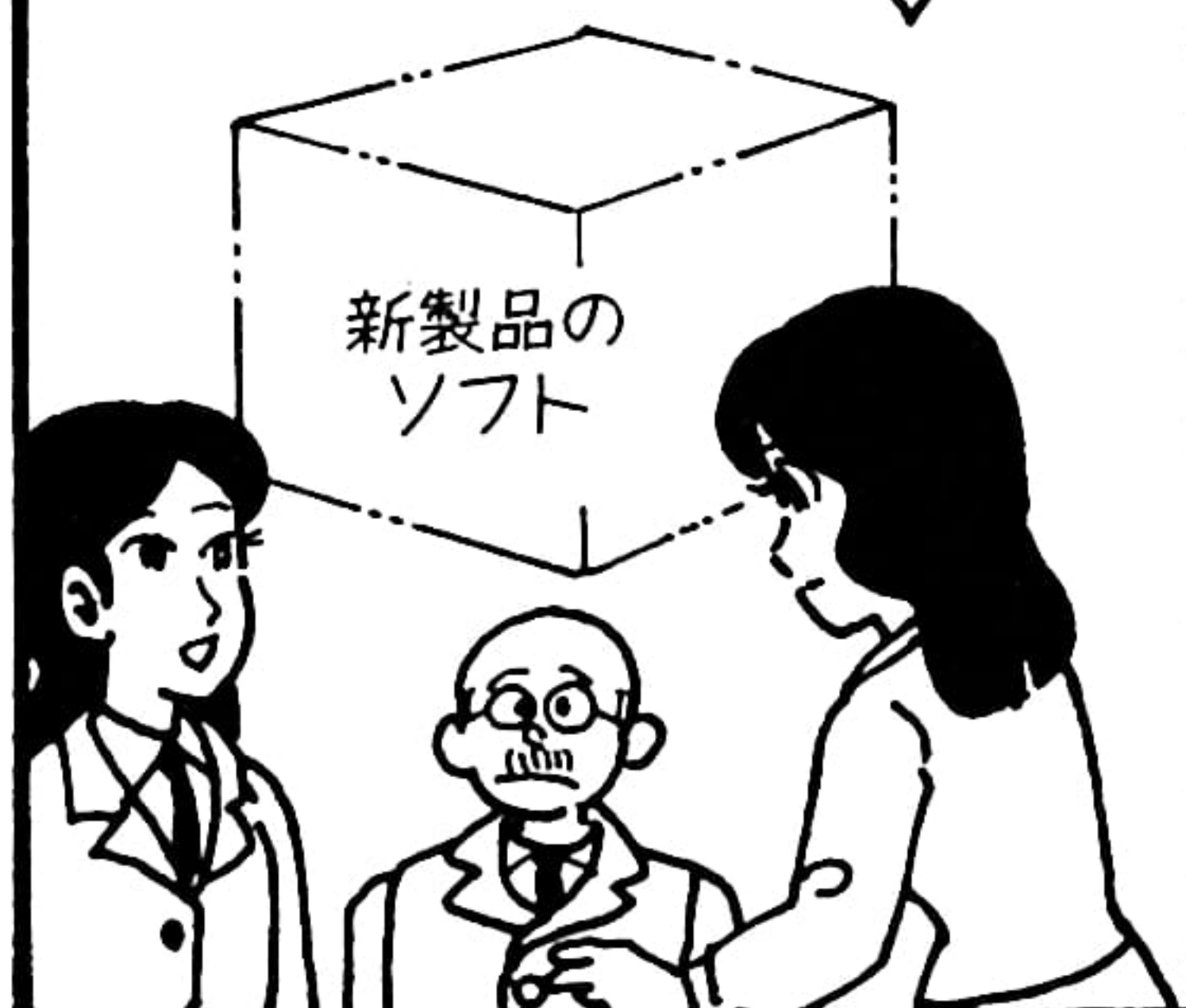
```
2000 57 28 34 18 39 18 00 00 00 00 00 00 00 00 00 00 W(4.9...
```

もし、このデバッガがなかったら…。このDDTはマシン語開発には不可欠です。

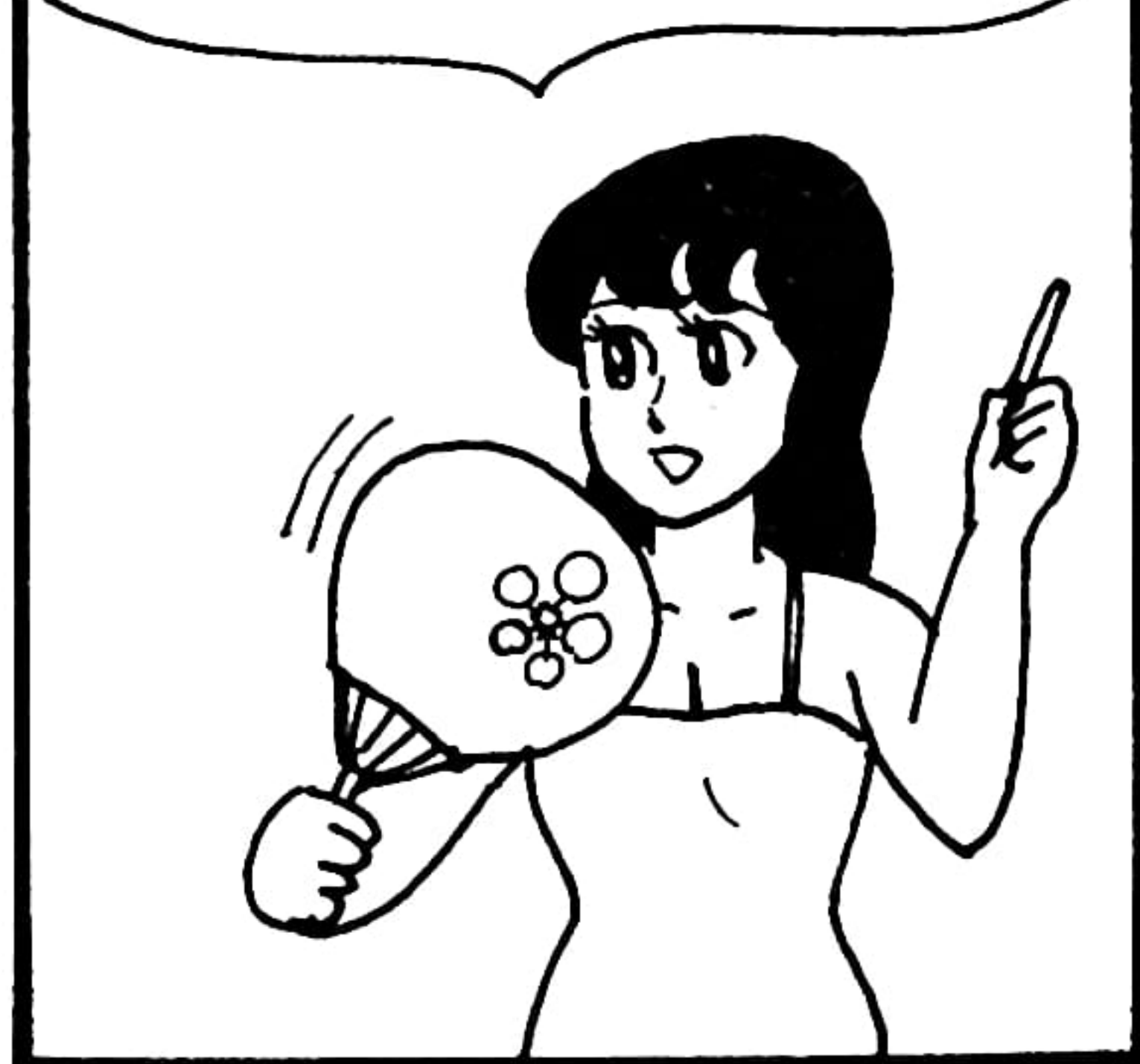
ここまでの画面表示をハードコピーしました。Xコマンドでは右はしがフラグの状態、その右どなりがAレジスタの内容、さらにその次がBC、DE、HLのレジスタペアです。Sはスタックポインタ、Pはプログラムカウンタです。前のプログラムカウンタの内容とニーモニックの関係を確認してください。

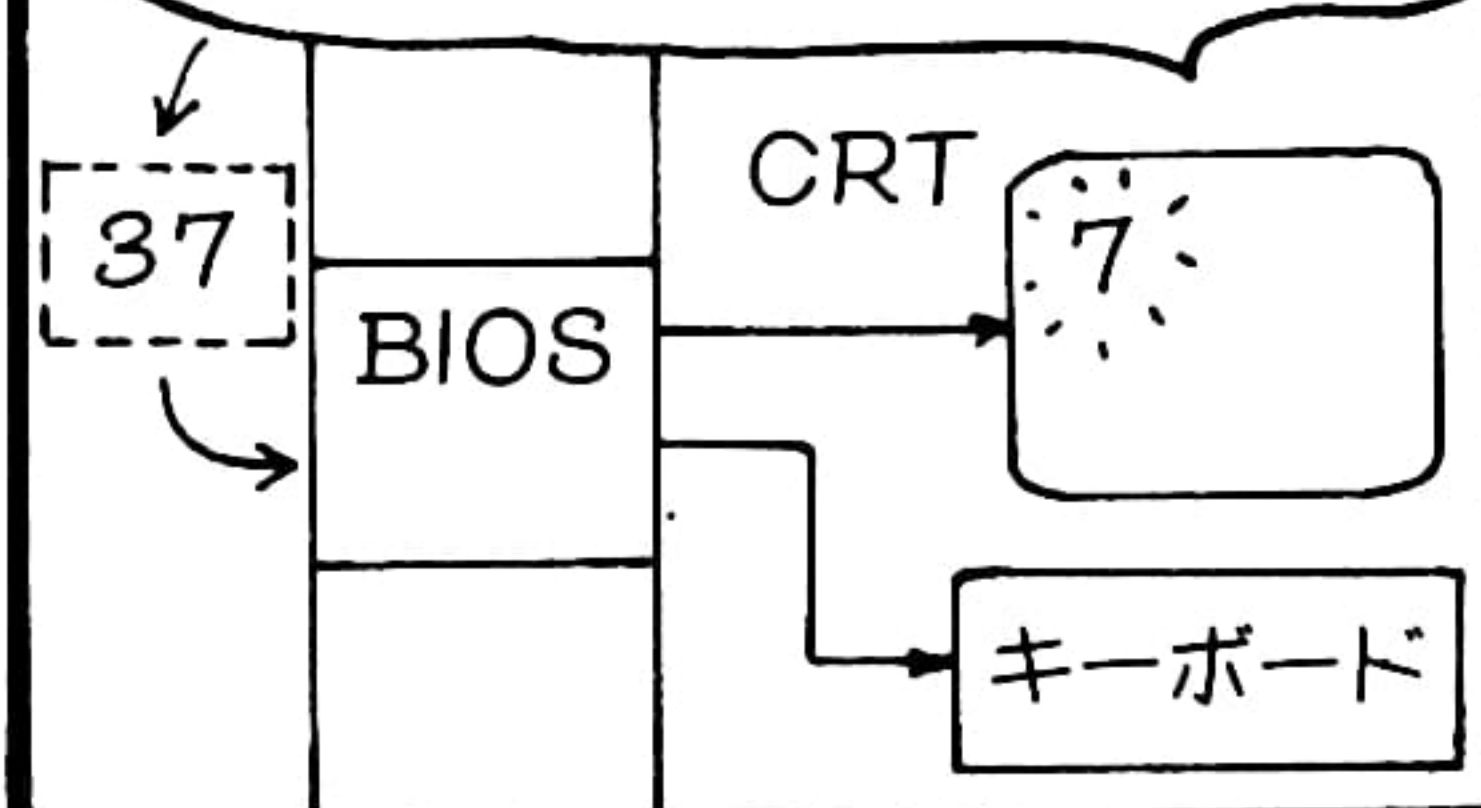
```
A>DDT TAKAKO.HEX
DDT VERS 2.2
NEXT  PC
0163 0000
-F2000,200F,0
-G100,114
*0114
-D2000,200F
2000 57 28 34 18 39 18 00 00 00 00 00 00 00 00 00 00 W(4.9..........
-G100,12A
*012A
-D2000,200F
2000 75 67 52 18 39 18 00 00 00 00 00 00 00 00 00 00 ugR.9..........
-X
C0Z1M0E0I0 A=52 B=2002 D=0000 H=2005 S=0100 P=012A LXI  B,2002
-G100,131
52 *0131
-D2000,200F
2000 75 67 52 18 39 18 00 00 00 00 00 00 00 00 00 00 ugR.9..........
-X
C0Z1M0E1I0 A=00 B=2002 D=0003 H=0000 S=0100 P=0131 DCX  B
-
-X
C0Z1M0E1I0 A=00 B=2002 D=0003 H=0000 S=0100 P=0131 DCX  B
-^C
A>
```

CP/Mは新しいハード構成のパソコンを作る時、またそのパソコン用のBASICインタプリタを作る時などに威力を発揮することになるでしょう。

新製品のソフト

実は私たちがBASICや簡易言語でパソコンを操作している時は、CP/Mは全然表に出てこないのです。

CP/M

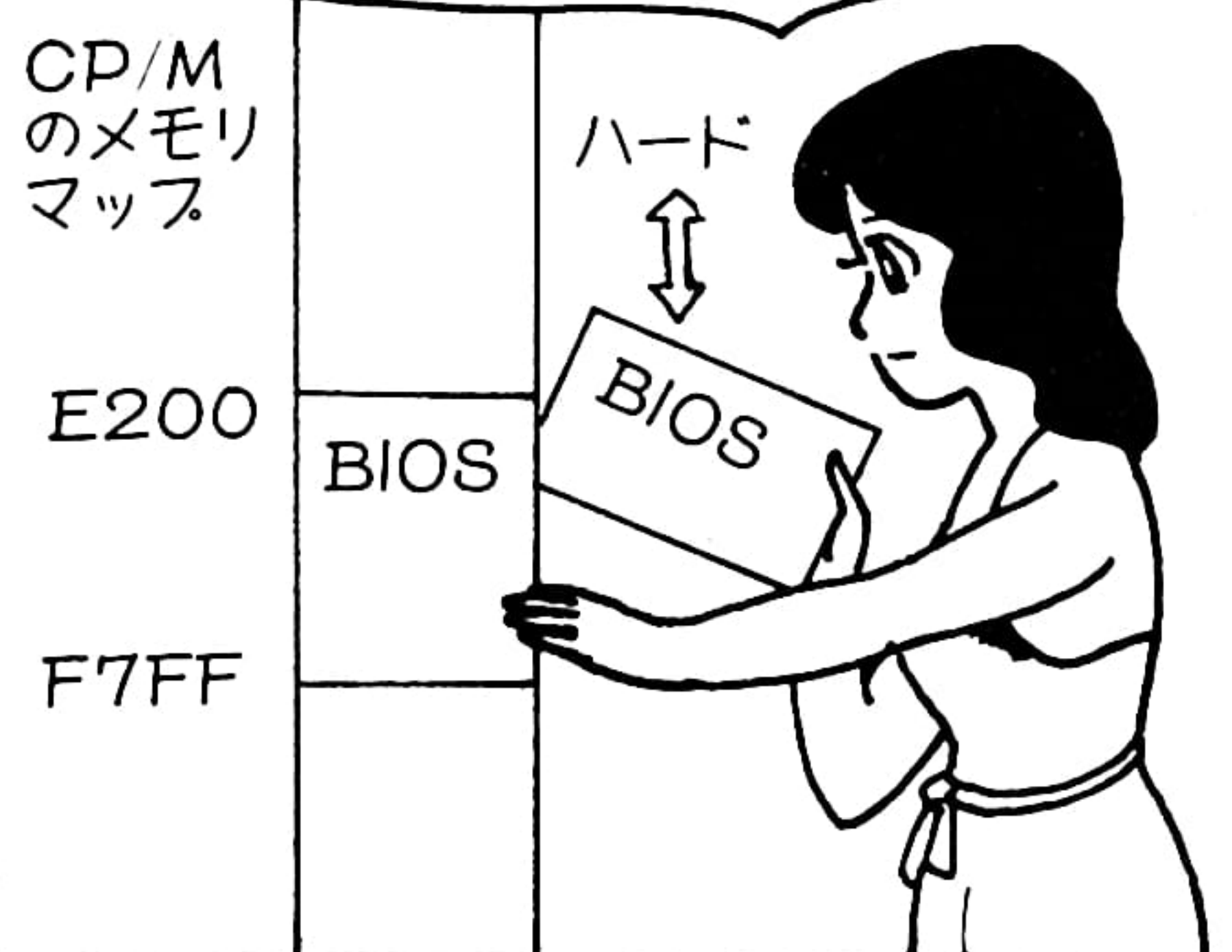
CP/M、CP/Mと騒いだわりにはそれらしいものがあまり出てきませんでした。いったいどこがCP/Mなのでしょう。

このシステムコールのファンクションの中にはI/Oのハードに関するプログラムはないのです。つまり、ハードの構成は本文では受け持たずアドレス5から始まるサブルーチンがハード選択の機能を持っていることがわかります。

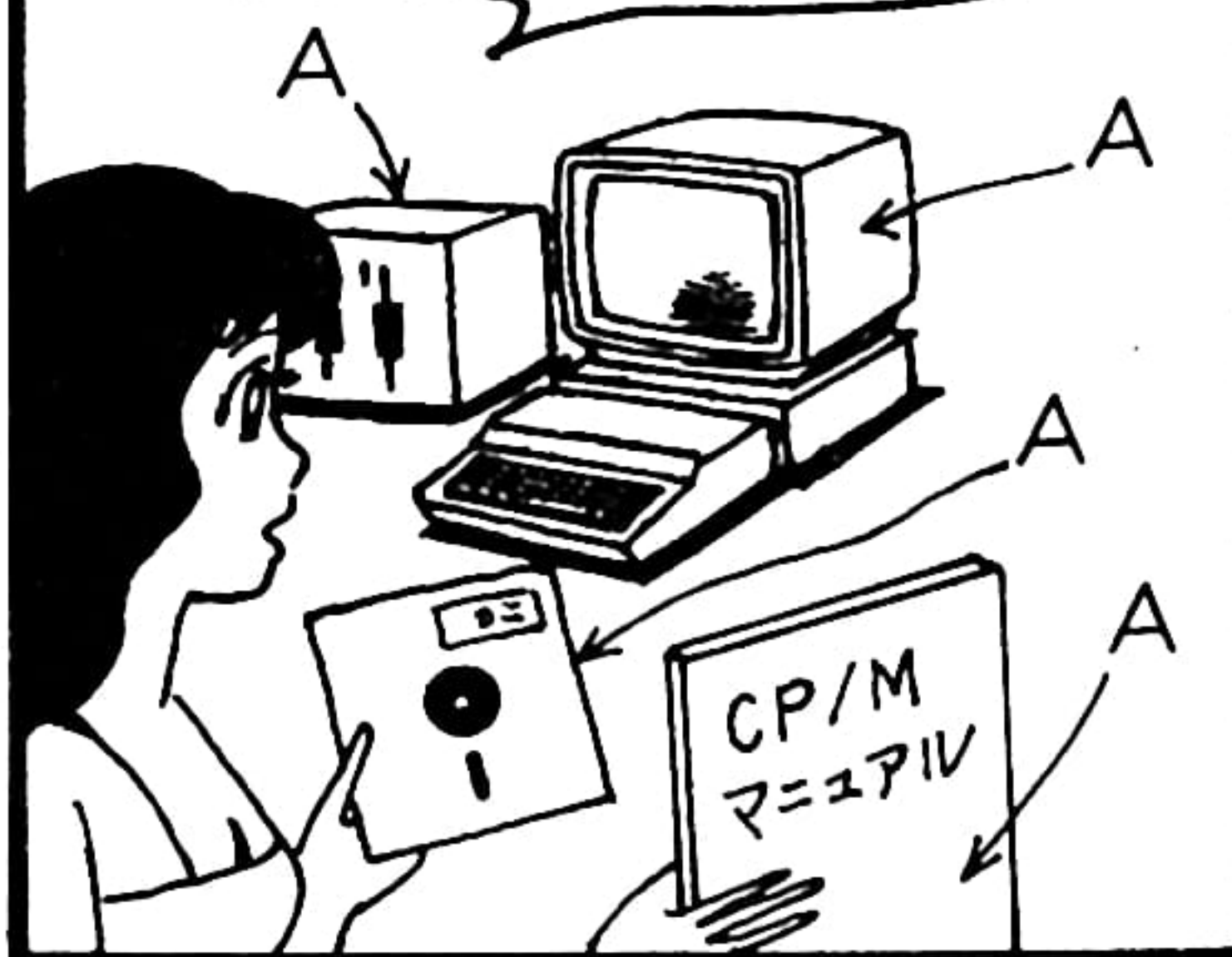
37
BIOS
CRT
7
キーボード

今のプログラムで1キャラクタを表示するのに、システムコールの2番を使いましたね。

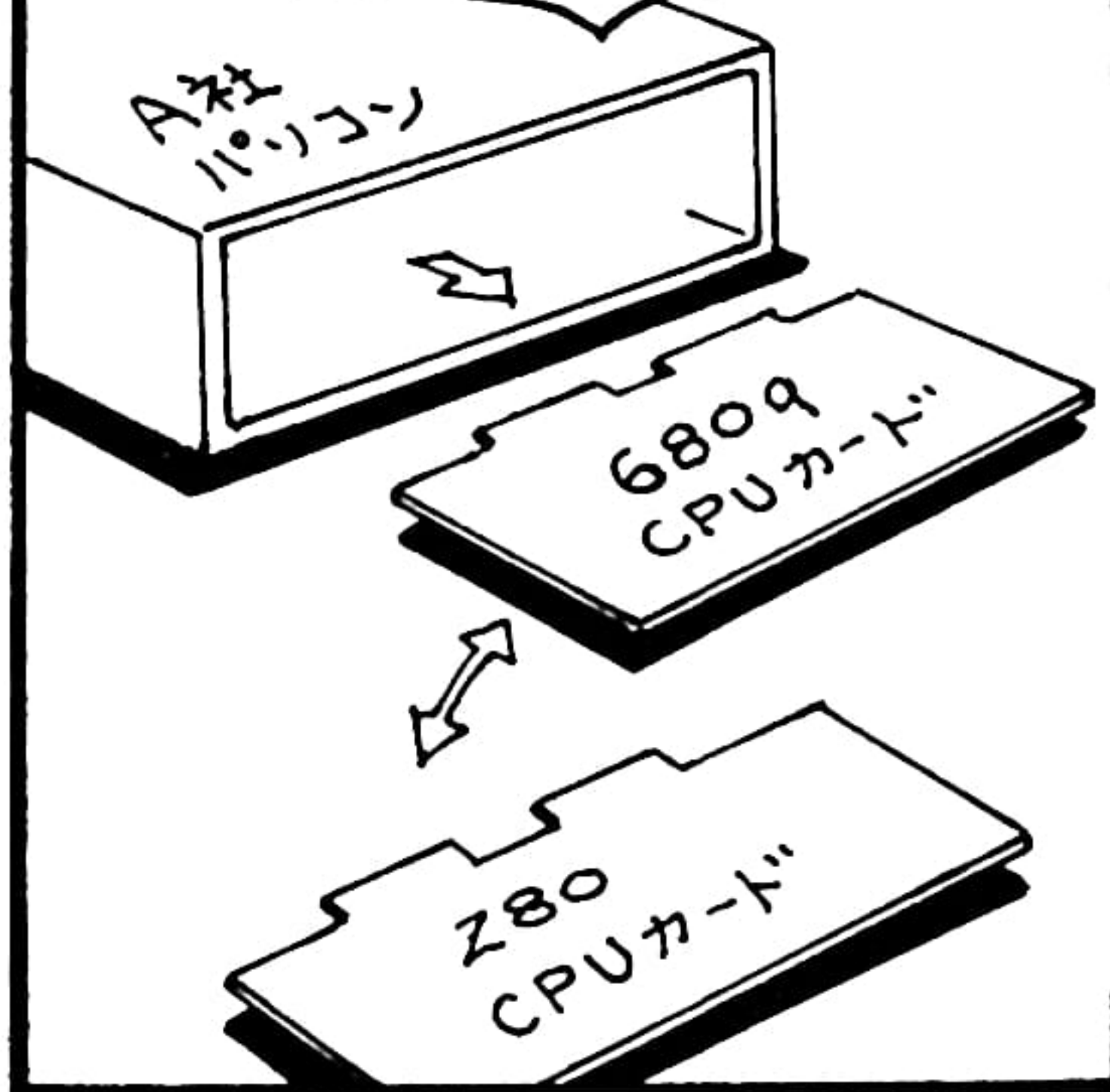
MVI␣C,2
MOV␣E,A
CALL␣5

7
37
E
キャラクタに直して表示する。

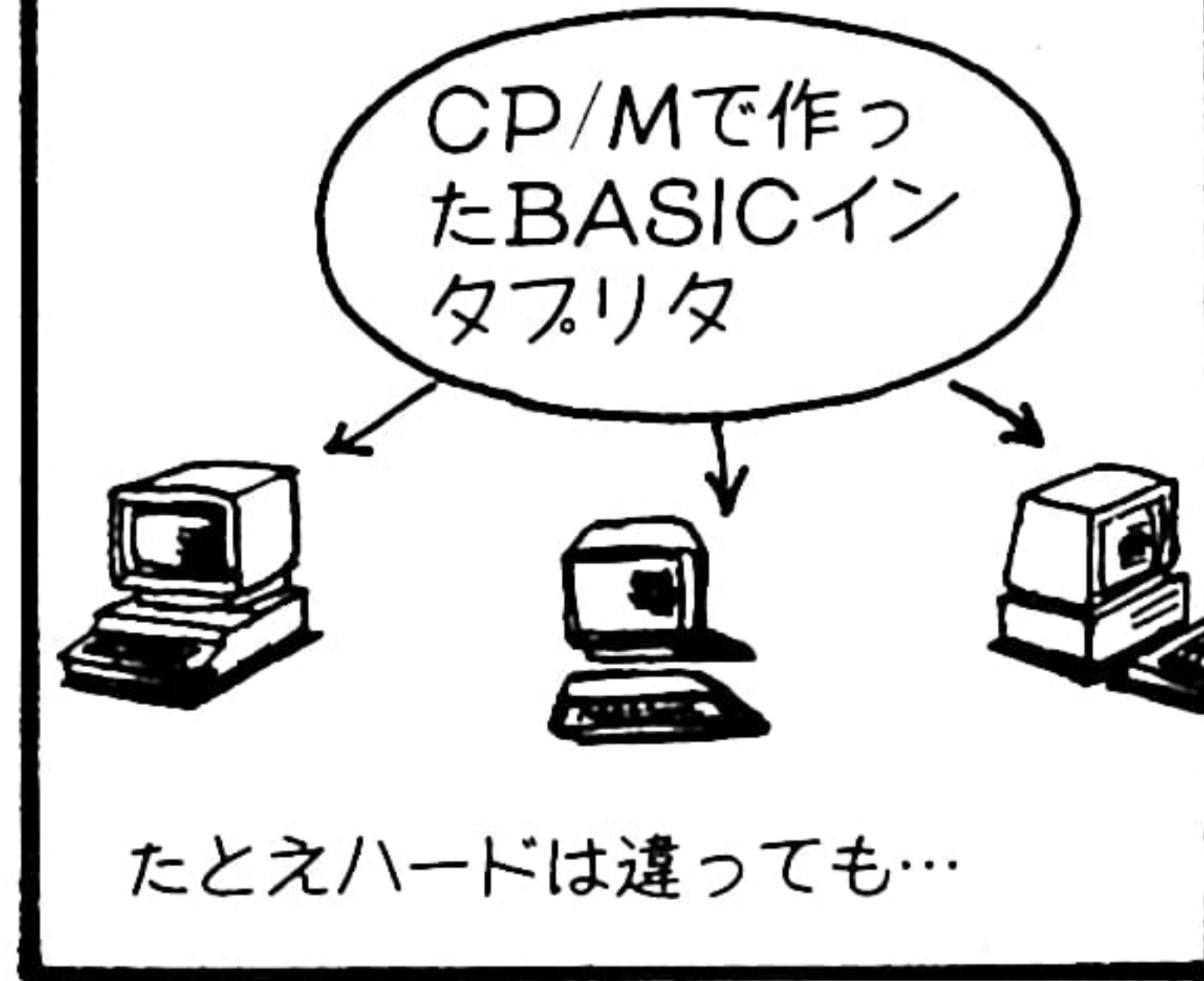
それまでに作ったインタプリタ自身は変えることなくCP/MのI/Oに関係のあるブロックだけを変更すればよいのです。

CP/Mのメモリマップ
ハード
E200
BIOS
BIOS
F7FF

またこの時注意が必要なのはA社のパソコンのA社のZ80カードと変換し、A社のそのパソコン用のCP/MでないとダメということですB社のZ80カードとC社のCP/Mでやろうとしても無理です。

A
A
A
A
CP/M マニュアル

また、80系以外のパソコンでCP/Mを走らせるためには、CPUカードをZ80のついたカードに変換する必要があります。

A社 パソコン
6809 CPUカード
Z80 CPUカード

つまり、同じCP/Mのうえで作られたマシン語は本来どのパソコンでも走らなければならないのですが、現実にはうまくいかないこともあるようです。

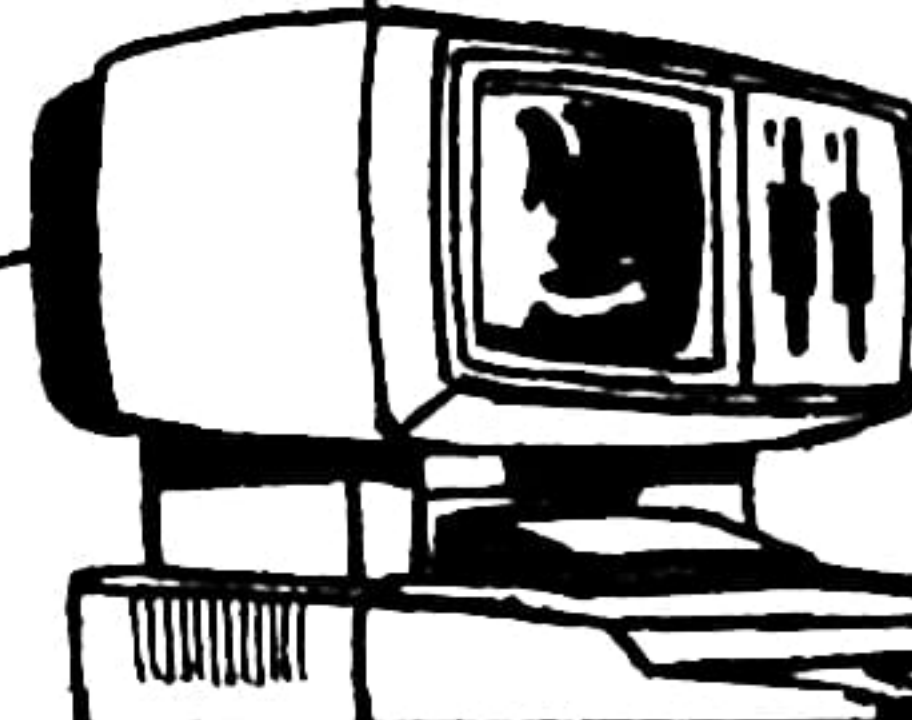
CP/Mで作ったBASICインタプリタ

たとえハードは違っても…

③Z80アセンブラを試す

Z80のアセンブラには少なくともM80とL80が必要です。システムディスクにコピーします。

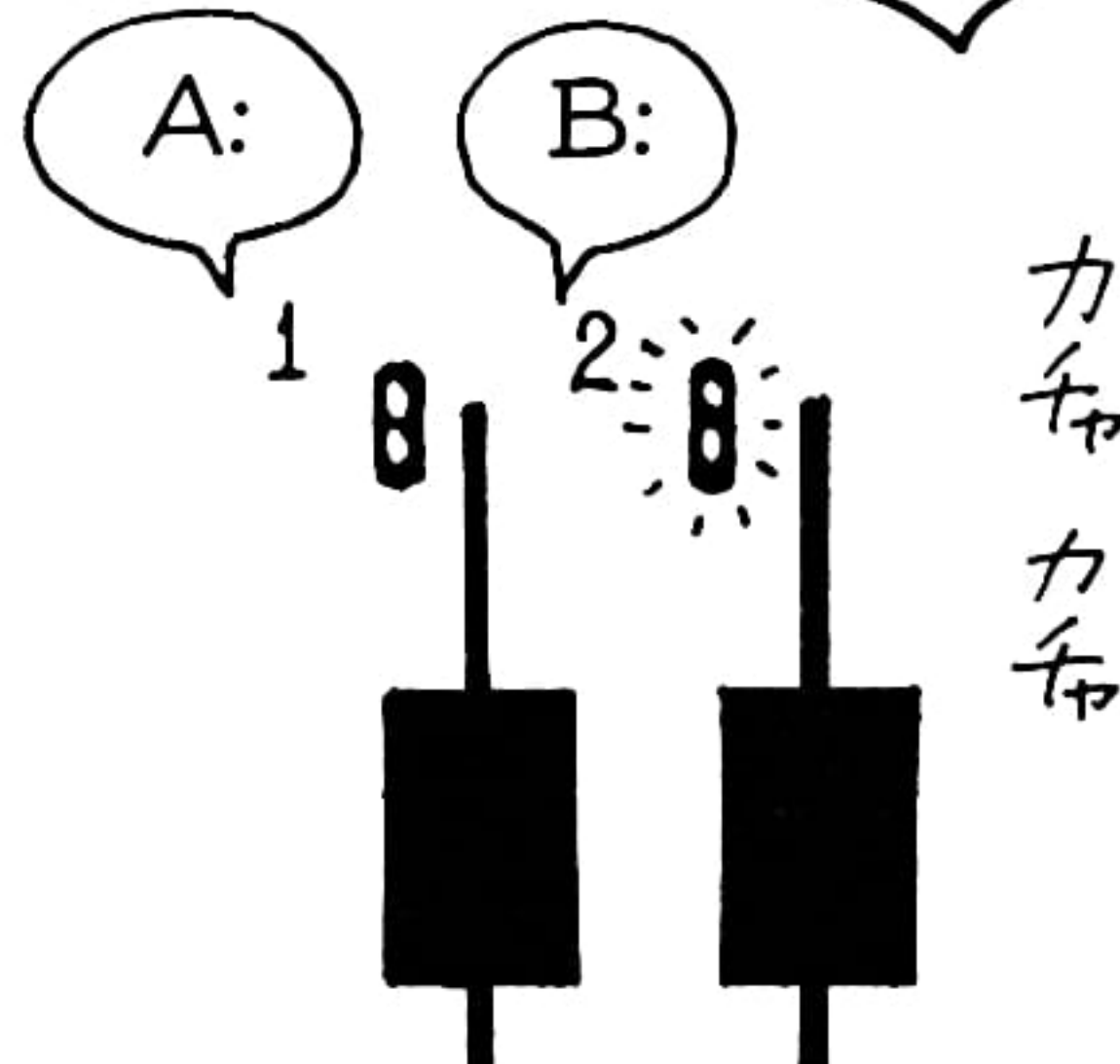
A>DIR
…………M80␣COM………
…………L80␣COM………
…………ZSID␣COM………

.MACファイルをつくります。

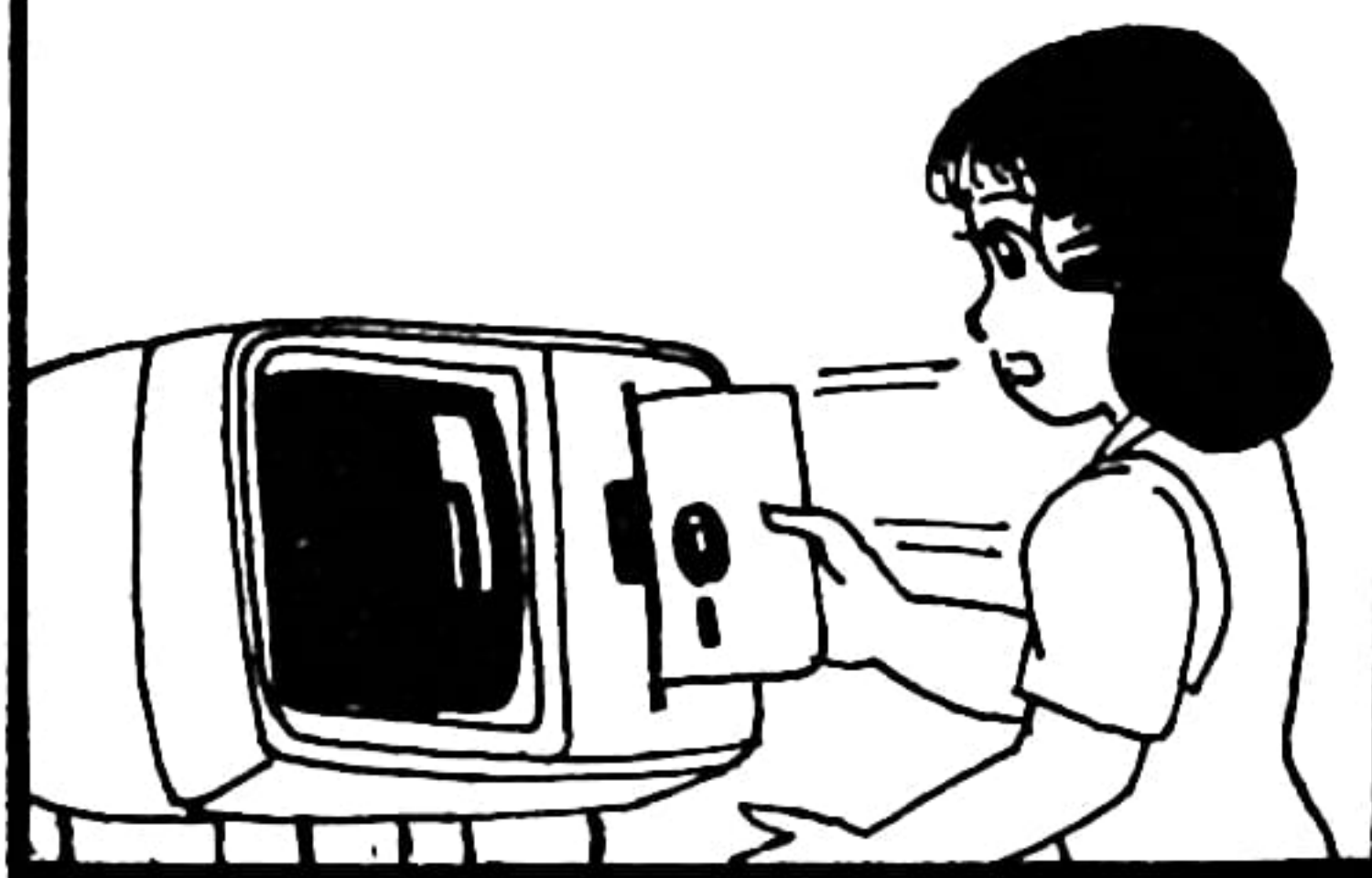
M80、L80などはオプションです。

L80.COMもZSID.COMも同様にコピーします。
PIP␣A:=B:80.COM↲
PIP␣A:=B:ZSID.COM↲

A>PIP␣A：=B：M80.COM
A>PIP␣A：=B：L80.COM
A>PIP␣A：=B：ZSID.COM

これでB:に入っているM80.COMファイルがA:のFDにコピーされます。

A:
B:
1
2
カチャ
カチャ

コピーはシステムFDをB:に入れ自分のをA:に入れます。自分のにもCP/Mのシステムが入っていなければなりません。
PIP␣A:=B:M80.COM↲

キーイン
表示
A>PIP␣A：=B：M80.COM

ソースファイルはエディタで作ります。まず、A>の状態にします。

前のと同じプログラムをZ80のニーモニックに直してソースファイルを作り、それをアセンブルしてみます。

エディタ以外は使用するプログラムや拡張子が異なります。

ファイル＼ニーモニック	8080	Z80
エディタ	ED.COM	ED.COM
アセンブラ	ASM.COM	M80.COM
ローダ	LOAD.COM	L80.COM
デバッガ	DDT.COM	ZSID.COM
拡張子	.ASM	.MAC

ここで少しめんどうなのですが、次のコマに書いているような内容をつけ加えたプログラムにする必要があります。
ASEG↲
ORG␣
0100H

```
  :*I
1 :ASEG
2 :ORG 0100H
3 :
```

次に、修正などを行なうなら＃A↲としてファイルをメモリに移しますが、この場合おニューなのでこのままIコマンドでプログラムを書き込んでいきましょう。I↲

```
NEW FILE
   :*I
  1:
```

次にED␣HITOMI.MAC↲とキーインします。HITOMIがこのファイルの名前です。必ず.MACをつけておかなければあとでアセンブルできません。

```
A>HITOMI.MAC
NEW FILE
   :*
```

.MACファイル作成時の注意事項

```
ASEG
ORG␣0100H
.PHASE␣0100H
.Z80
ENTRY␣TOMOKO,
        LOOP,
        DONE,
        SATOMI

TOMOKO:LD␣HL,2000H
LD␣(HL),57H
  .
  .
  .
  .
  .
  .
POP␣BC
RET
.DEPHASE
END␣TOMOKO
```

ロケーションカウンタをゼロに戻す。

.PHASE␣nn 〜 .DEPHASEの外になければERRORになった。

ここまでがアセンブラの対象となる。

Z80のニーモニックであることを示す。これがないとM80の時にそれなりの操作が必要。

Z80のニーモニックには.MACという拡張子をつけます！

本文は次のページを参照してください。

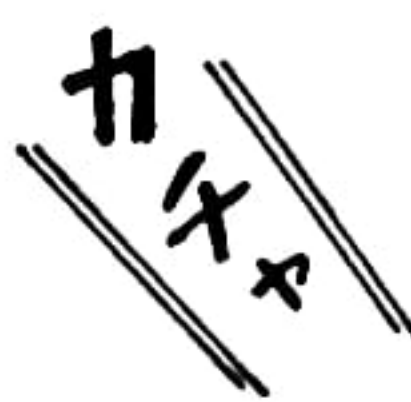
使用するラベルを列記しておく。

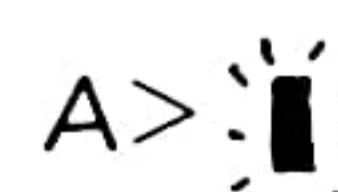
最初のニーモニックにラベルをつけこれをEND文の後につける。これがないとL80の時ERRORとなる。

アセンブルファイルの作成とリストアップのプリントをするコマンドです。HITOMI,LST:=HITOMI↵

A>M80
*HITOMI,LST：=HITOMI
A>▮
ビィーッ ビィーッ

HITOMI.MACはTAKAKO.ASMのニーモニックを変えただけですがそれでもエラーはありました。これは修正完了後のHITOMI.MACです。アセンブルはM80↵

A>M80
*
カチャ

エディタの終わりはコマンドE。これでFDにHITOMI.MACファイルが作成されます。E↵

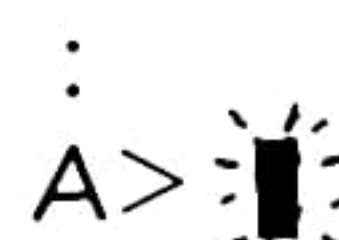
71：　*
71：END␣TOMOKO
72：　*E
A>▮
カチャ カチャ

ではこのプログラムを走らせましょう。HITOMI↵OKです。

A>HITOMI
52 67 75

ディレクトリコマンドでHITOMIファイルを見てみましょう。

A>DIR HITOMI.*
A: HITOMI　BAK
: HITOMI　MAC : HITOMI
: HITOMI　REL
A: HITOMI　COM

続いてリンカーにかけます。L80↵続いてのコマンドはHITOMI/E,HITOMI/N↵です。これで.COMファイルができます。

A>L80
Link-80 Release 3·3
*HITOMI/E,HITOMI/N
:
A>▮

```
                         ;FILE NAME
                         ;HITOMI.MAC
0000'                    ASEG;Z80 ASM.
                         ORG 0100H
                         .PHASE 0100H;Z80 ASM
                         .Z80
                         ENTRY TOMOKO,LOOP,DONE,SATOMI
0100      21 2000        TOMOKO:LD HL,2000H
0103      36 57          LD (HL),57H
0105      23             INC HL
0106      36 28          LD (HL),28H
0108      23             INC HL
0109      36 34          LD (HL),34H
010B      23             INC HL
010C      36 18          LD (HL),18H
010E      23             INC HL
010F      36 39          LD (HL),39H
0111      23             INC HL
0112      36 18          LD (HL),18H
0114      01 2000        LD BC,2000H
0117      21 2003        LD HL,2003H
011A      AF             XOR A
011B      1E 03          LD E,03H
011D      0A             LOOP:LD A,(BC)
011E      8E             ADC A,(HL)
011F      27             DAA
0120      02             LD (BC),A
0121      1D             DEC E
0122      CA 012A        JP Z,DONE
0125      03             INC BC
0126      23             INC HL
0127      C3 011D        JP LOOP
012A      01 2002        DONE:LD BC,2002H
012D      0A             LD A,(BC)
012E      CD 013C        CALL SATOMI
0131      0B             DEC BC
0132      0A             LD A,(BC)
0133      CD 013C        CALL SATOMI
0136      0B             DEC BC
0137      0A             LD A,(BC)
0138      CD 013C        CALL SATOMI
013B      C9             RET;RETURN TO CP/M
013C      E6 F0          SATOMI:AND 0F0H
013E      0F             RRCA
013F      0F             RRCA
0140      0F             RRCA
0141      0F             RRCA
0142      E6 0F          AND 0FH
0144      F6 30          OR 30H
0146      C5             PUSH BC
0147      0E 02          LD C,2
0149      5F             LD E,A
014A      CD 0005        CALL 5
014D      C1             POP BC
014E      0A             LD A,(BC)
014F      E6 0F          AND 0FH
0151      F6 30          OR 30H
0153      C5             PUSH BC
0154      0E 02          LD C,2
0156      5F             LD E,A
0157      CD 0005        CALL 5
015A      0E 02          LD C,2
015C      1E 20          LD E,20H
015E      CD 0005        CALL 5
0161      C1             POP BC
0162      C9             RET;RETURN TO HITOMI
                         .DEPHASE;Z80 ASM
                         END TOMOKO
```

```
␣␣␣␣␣␣JP␣LOOP
DONE:LD␣BC,2002H
␣␣␣␣␣␣LD␣A,(BC)
```

このようなソースプログラムの方が見やすいことは確かです。アセンブルには関係ありませんが。

普通はリストを見やすくするために、ラベルとニーモニックは TAB キーを使ってずらせます。

これがZ80のニーモニックで作った.RELファイルです。おや？　マシン語のアドレスが変ですね。これはリストが見やすいようにしているだけです。

LD␣BC,2000H

↓

01 20 00

あれれ？

ご心配なく。マシン語はちゃんと

01 00 20　です。

エディタでソースプログラムをキーインしながら感じたことは、とても使いやすいニーモニックだなということです。たとえば8080の LDAX␣B よりもZ80の LD␣A,(BC) の方がラクラクとキーインできます。

ロード LD␣A,(BC)

A

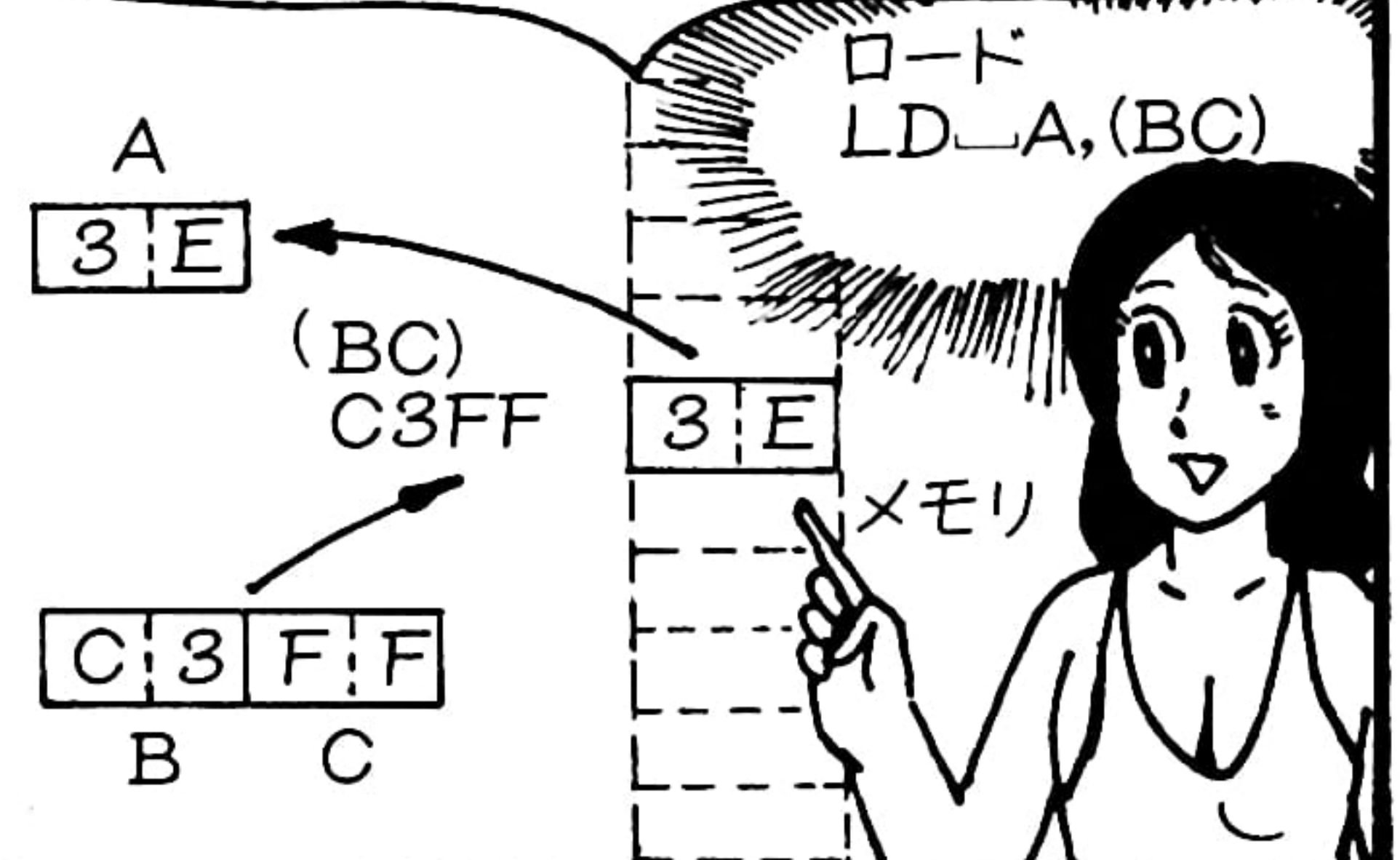
3E

(BC) C3FF

3E

メモリ

C3FF

B C

CPUは8080や8085を使っていてもニーモニックはこのZ80のものを使うことが多いようです。以上、簡単ですが、これでZ80ニーモニックのアセンブラの話はオシマイです。

Ⓔ RS-232Cでデータ転送

こういう通信のためのコマンドが各パソコンに用意されています。

音響カプラ

普通の電話機

パソコン同士のデータのやりとりはBASICで簡単にできます。ここではパソコンとポケコンをつないでみました。アスキーコードを理解するのがカギです。

4～5万円で買えます。

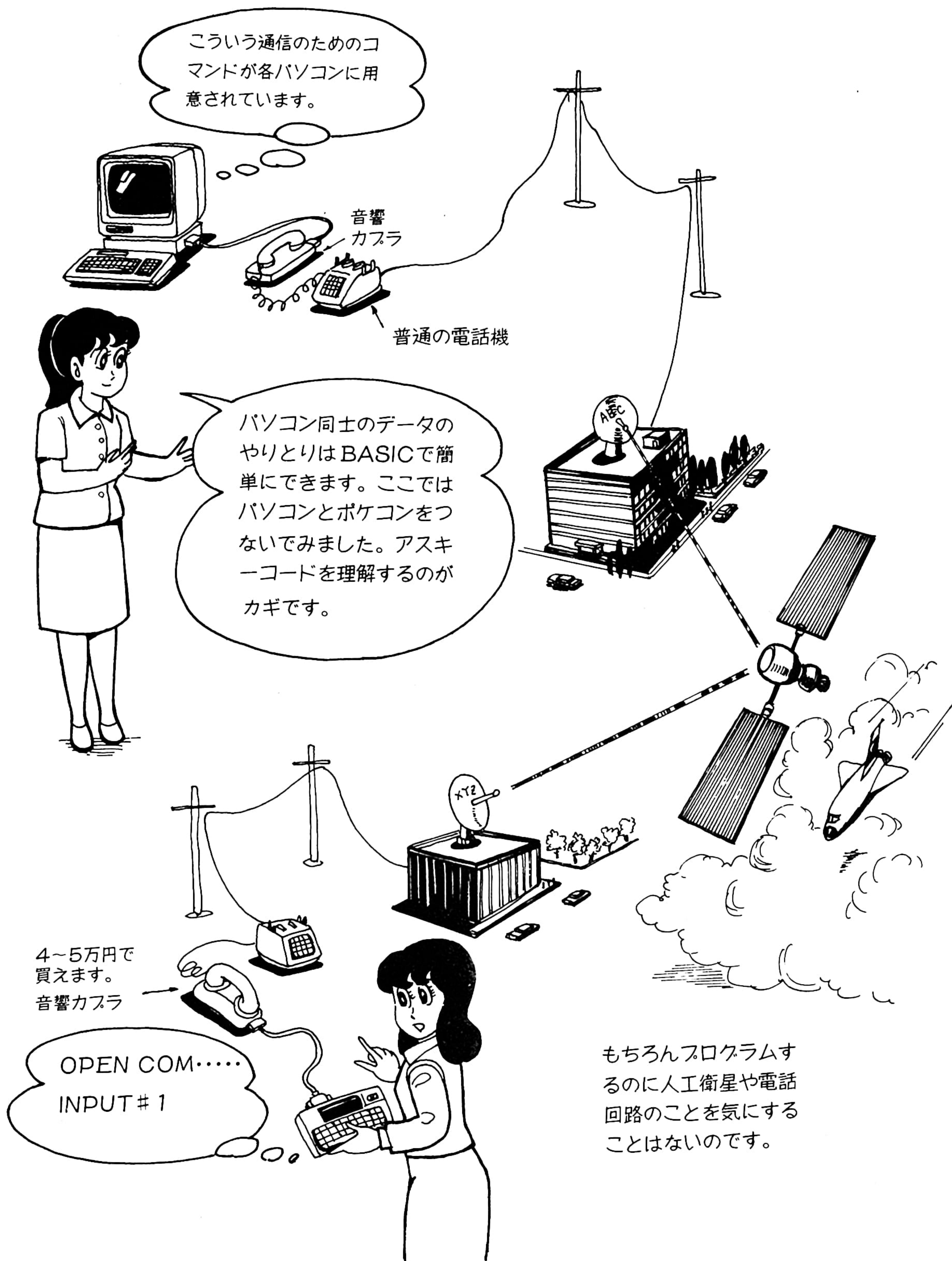
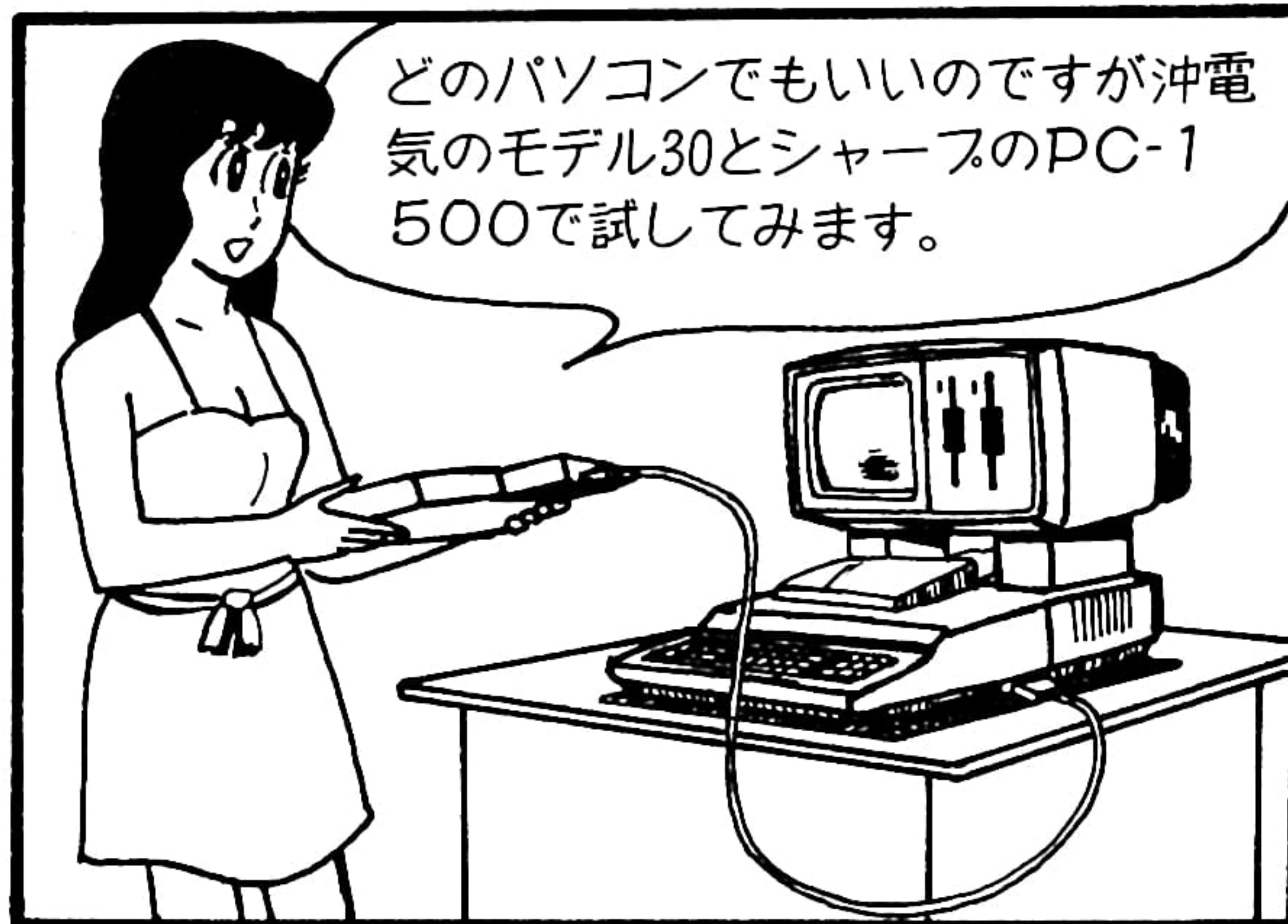
音響カプラ

OPEN COM……

INPUT＃1

もちろんプログラムするのに人工衛星や電話回路のことを気にすることはないのです。

①RS-232C活用のハード

どのパソコンでもいいのですが沖電気のモデル30とシャープのPC-1500で試してみます。

RS-232Cはプリンタなどの接続ポートとしても使われますが…

これを使ってデータ通信をやってみます。

このデータ転送方法は、同期信号がない非同期式です。マイコンの内部でも、データラッチのための制御信号が出ていましたがそれがないのです。では、どうやってデータを正しく受け取ることができるのでしょうか。

データ

ラッチ信号

プログラムを作る時にはあまり必要ありませんが、一応どんな方法でハード的につなぐのかを知っておくのもよいと思います。いろいろと応用が効きますからね。

データを取り込むタイミングは1/2ビット時間長だけずらしてあるので、1ビット時間が長い場合は信頼性が高いといえます。

1
転送信号
0

取込みデータ　スタート　1　0　0　1

最初に'0'になってから1/2ビット時間長までこの状態が続いた時にこれをスタートビットと認識します。そして、これ以後の1ビット時間ごとに取り込む内容がデータとなります。

転送信号 1 0

サンプリング

1/2 ビット時間

最初のスタートビット検出のために、ビット時間の何分の1かの短いタイミングでサンプリングしているのがミソです。「信号なし」は'1'の状態を保っているとします。受信側では、サンプリングを続け、

転送信号 1 0

サンプリング

サンプリング続行

これがどのような意味のコードなのかは、送受信両側のプログラムで決まります。

送信側から下図のような波形が送られてきた時、受信側では36、F7と受け取ります。

1
0

スタート　0 1 1 0 1 1 0 0 1　ストップ　スタート　1 1 1 0 1 1 1 1 0　ストップ
D0 D1 D2 D3 D4 D5 D6 D7 パリティ　D0 D1 D2 D3 D4 D5 D6 D7 パリティ

サンプリング　サンプリング

データの次にパリティビット、その後に1ビット時間以上の'1'状態を保つストップビットが続きます。これで受信側は次のスタートビット検出のためのサンプリングに入るというわけです。

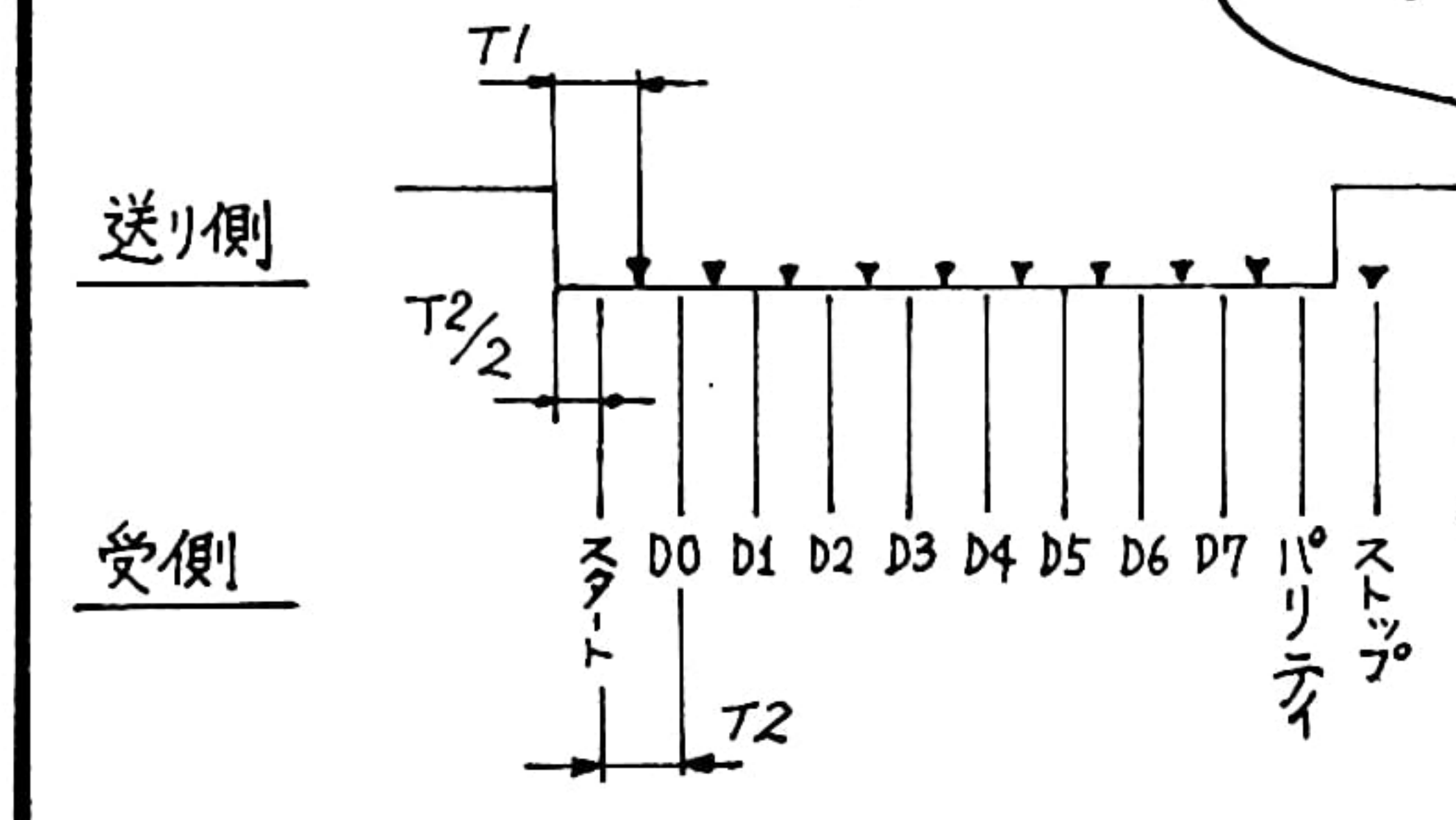
転送信号

取込みデータ　0　1　パリティ　ストップビット

サンプリング

そこで、このデータ転送方式は調歩同期式と呼ばれます。

送信側と受信側の歩調が合ってないとダメということです。

T1
送り側
T2/2
受側
スタート D0 D1 D2 D3 D4 D5 D6 D7 パリティ ストップ
T2

最初のスタートビットの検出だけでは、こううまくはいきません。1ビット時間、1コードのデータ長(ビット数)、パリティの有無、ストップビット数、などの条件を合わせておかねばならないのです。

たとえばRはアスキーコードで、10進表現では82、16進では52、2進では01010010となります。ポートから出ていくのは2進の電圧レベルの信号です。

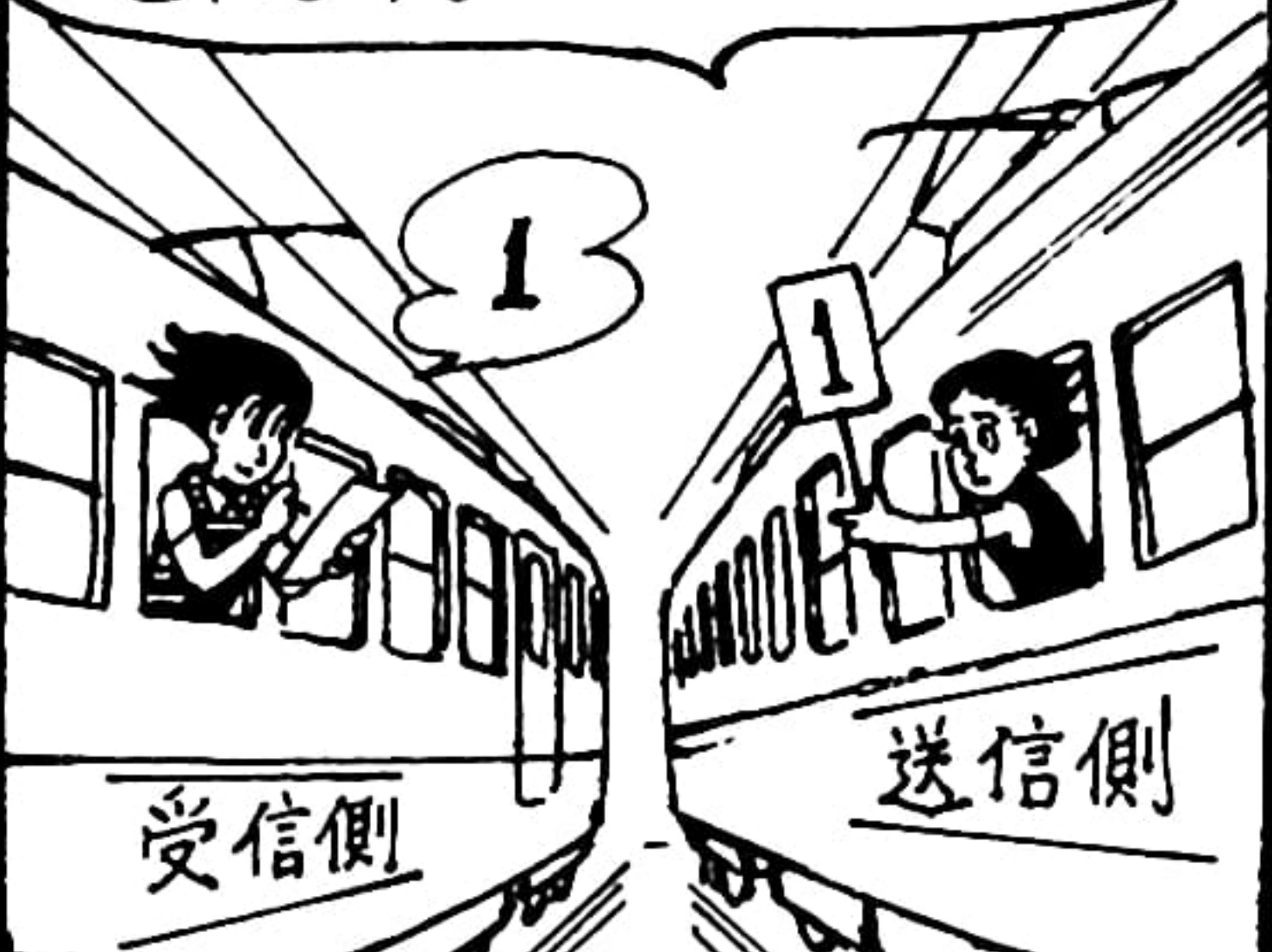
R
‖
01010010

どうして？
ね、どうして？

次に、データのコードを何ビットで表現するかを決めます。ふつうアスキー(ASCII)コードであれば7ビット、これにカナの入るJISコードであれば8ビット必要です。

0101 0010 (B)
5 2 (H)
82 (D)
R (ASCII)

まず送り側のビット時間T1と受信側のデータ取込み時間間隔T2が等しくなくてはなりません。この単位はボー(ビット/秒)で示されます。

1
受信側
送信側

受信側でこれを奇数パリティとして受け取った時、各ビットをたし算し、パリティビットまでの合計の最小桁が常に1であれば合格です。

1+0+0+1+
0+0+1+1+
1 ⇒ ‥1
P
100100111

奇数パリティというのはデータ8ビットの中に含まれる1とパリティビットの合計が奇数になるようにパリティビットを0か1にセットするものです。

パリティビット
0100 1001 [ ]
奇数パリティのとき 0 を入れる
偶数パリティのとき 1 を入れる

その次のパリティはビット検査方法のひとつで奇数パリティと偶数パリティの2種類あります。パリティビットをつけない場合もあります。

送 0100 1001 (49H)
反転 ノイズ
受 0110 1001 (69H)

もし、アスキー堀を埋められたら、すぐにも落城するだろう。

RS-232Cを自在に使いこなすには、まずアスキーコードの正体をはっきりつかんでおくことが必要だと思います。

RS-232C
アスキー堀 (ASCII)

ストップビットも1、2、3のいずれかをセットしておきます。調歩同期式ではこれらを同じ条件にセットしておかねばなりません。

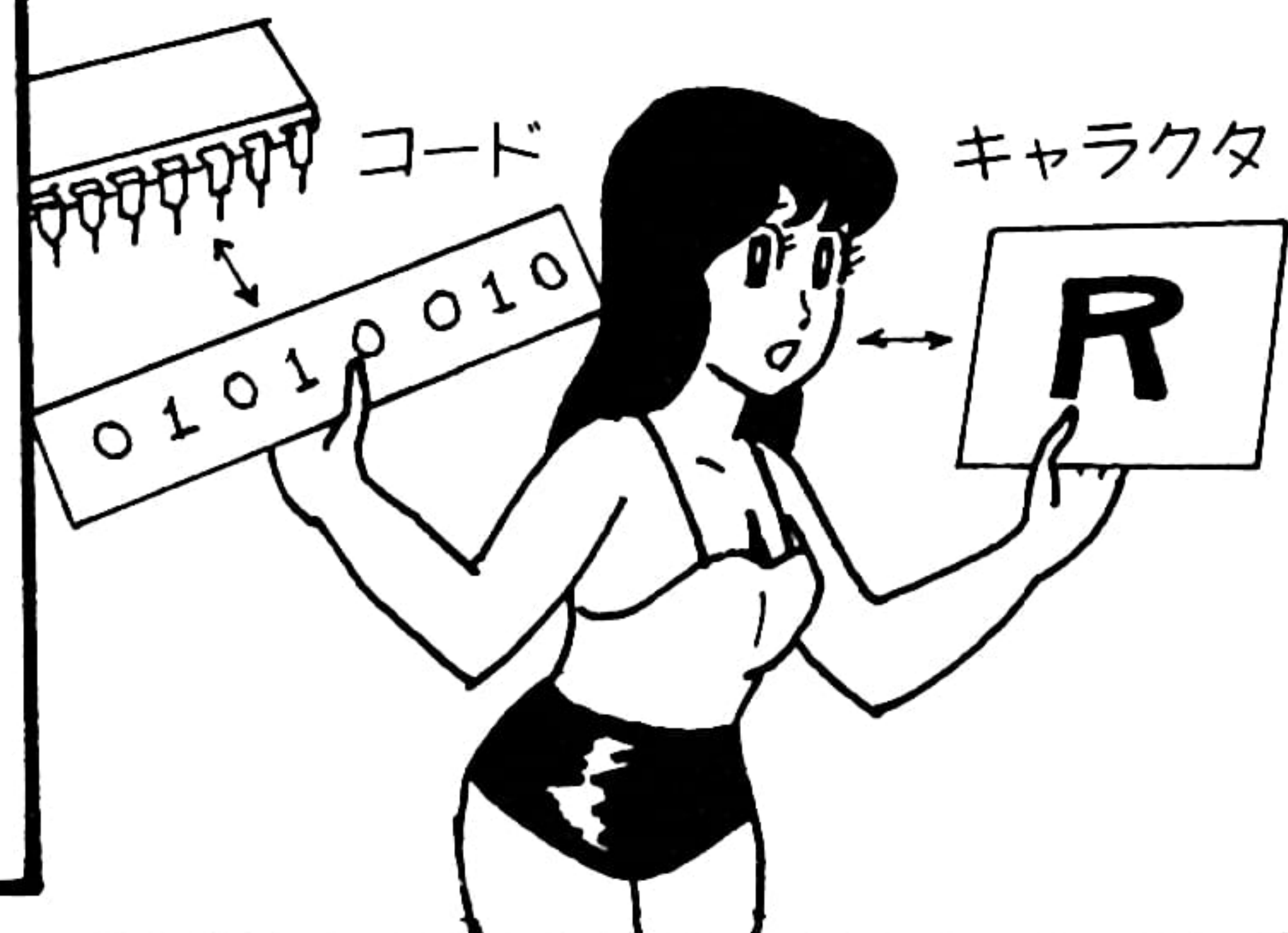
時間の間隔を等しくする！

②アスキーコードがカギ

CPUのマシン語とゴチャまぜにしないでくださいネ。CRT上の表示やキーボードの表示はキャラクタ、それを7ビットのコードに割り当てたものがアスキーコードです。

CPUのマシン語　全然別のもの　アスキーコード

コード　キャラクタ

01010010　R

7ビットでの組み合せは128通りあり、アスキーで使う文字や記号はこれで足りていました。

0123 4567

0 1 2 3 ⋮ D E F

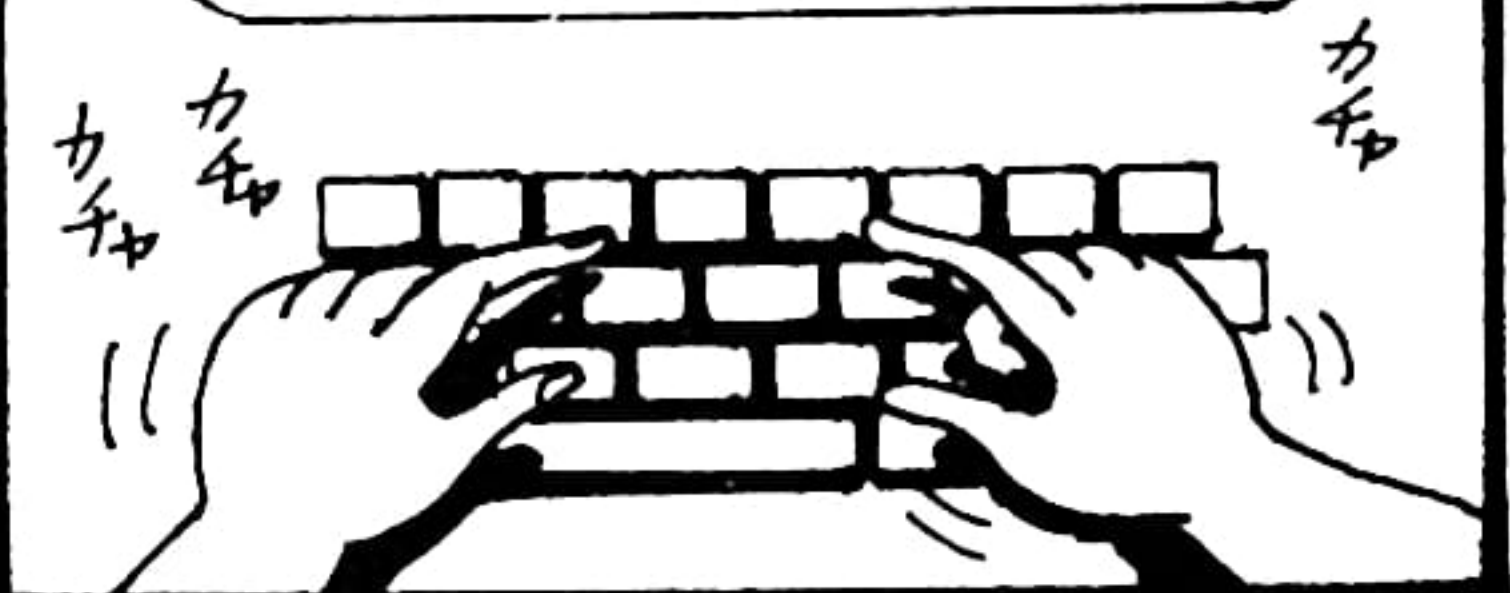
128種の文字を割り当てることができる。

そこで私たちの使う文字記号とCPUが使う時に必要なコードとの間に関連を持たせなければなりません。このコード変換で一番広く普及しているのがアスキー(ASCII)コードです。

アスキーコード	文字
0100 0001	A
0100 0010	B
0100 0011	C
0100 0100	D

私たちが使うのは、0～9までの数字、A～Zまでのアルファベット、それに+、−、(、＊、＝などの特殊文字です。(アメリカの人はこれで足りるわけです)

ABCDEFGHIJKLMNOPQR
STUVWXYZ
abcdefghijklmnopqr
stuvwxyz
1234567890

カチャ　カチャ　カチャ

8ビットでは7ビットよりもさらに128多くコードを割り当てることができます。

	01234567	89AB CDEF
0 1 2 3 ⋮ C D E F	ASCIIコード (128)	カタカナコード (128)

カタカナを入れた8ビットコードは「アスキーコード」ではなく「JISコード」になります。しかし、7ビット分はアスキーに準じたコードにしてあります。この部分もJISで独自に決めたのでは、コードに国際的な互換性がなくなり不利になるのは日本です。

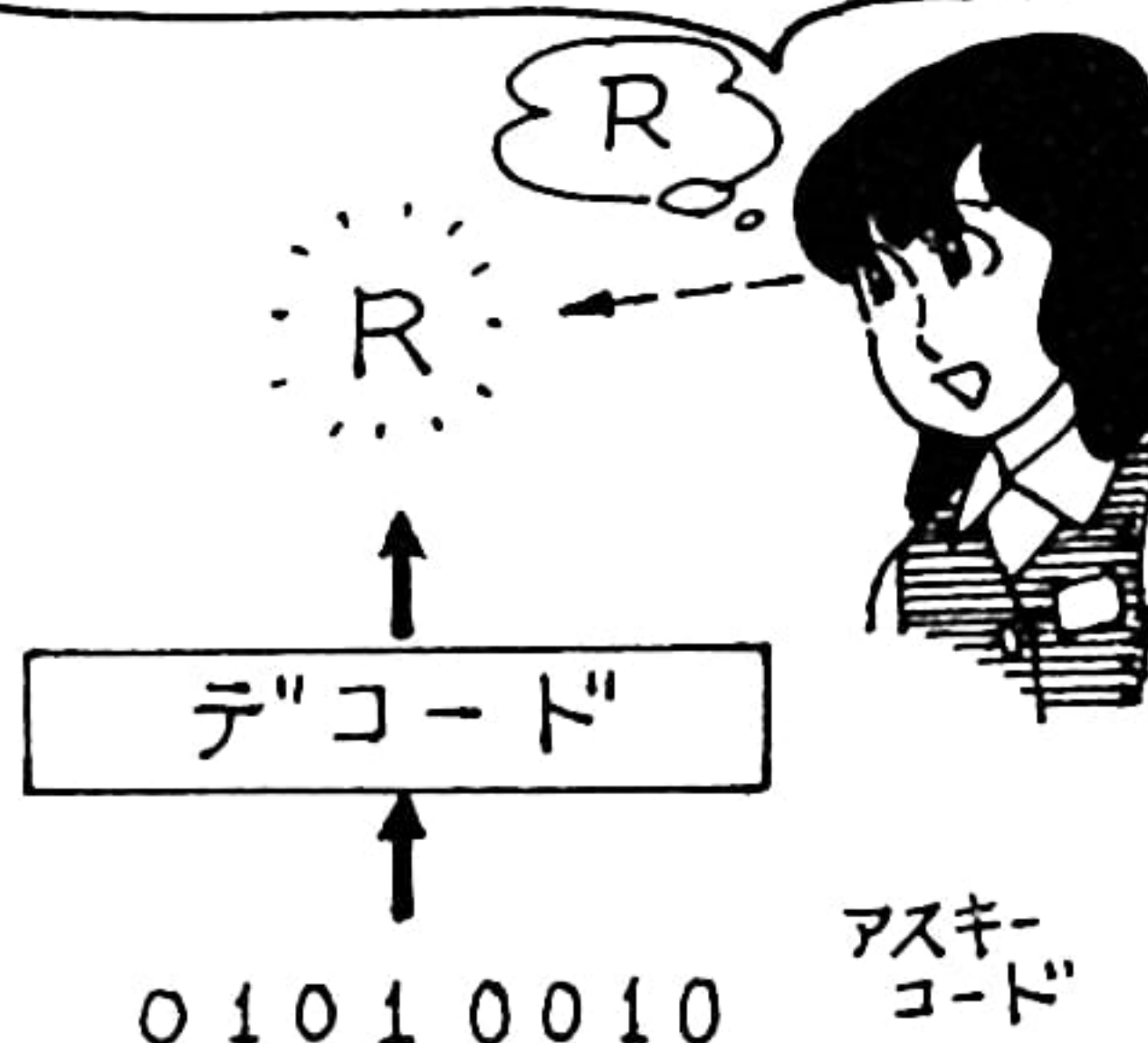
ASCII　カタカナ
7ビット
8ビット

ところが日本人の場合は、ひらがな、漢字、カタカナも合せて使いますので128のアスキーコードだけでは不便です。そこで、せめてカタカナだけでもということで、8ビットにし、増えた128コードにカタカナを割り当てたのです。

I LOVE JAPAN
ニッポン　ダイスキ

電子回路側からのアウトプットの時はアスキーコードをデコードし、文字記号に戻さねばなりません。

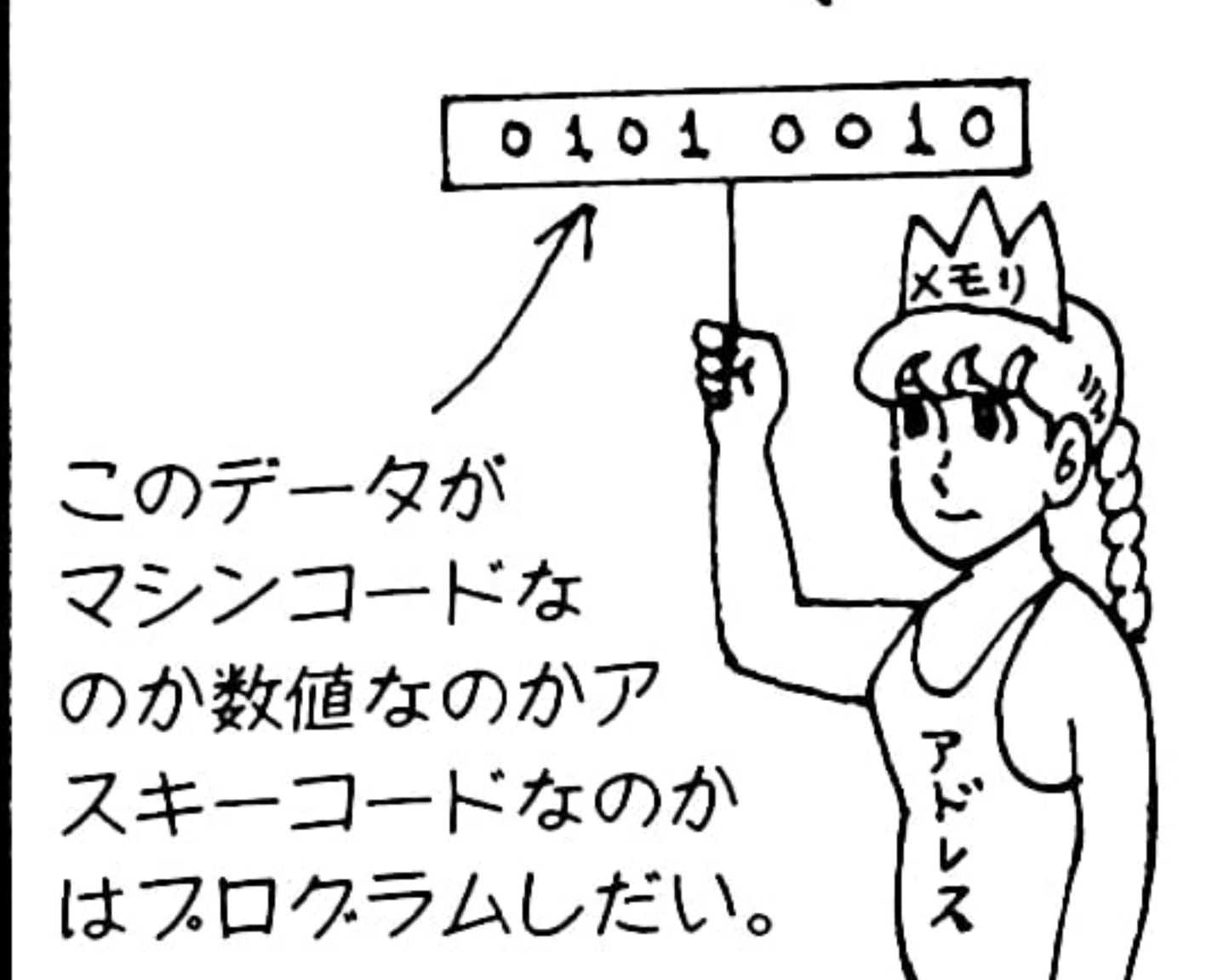
R

デコード

01010010　アスキーコード

電子回路でのデータの取扱は、文字記号の場合このアスキーコードで行なっているわけです。

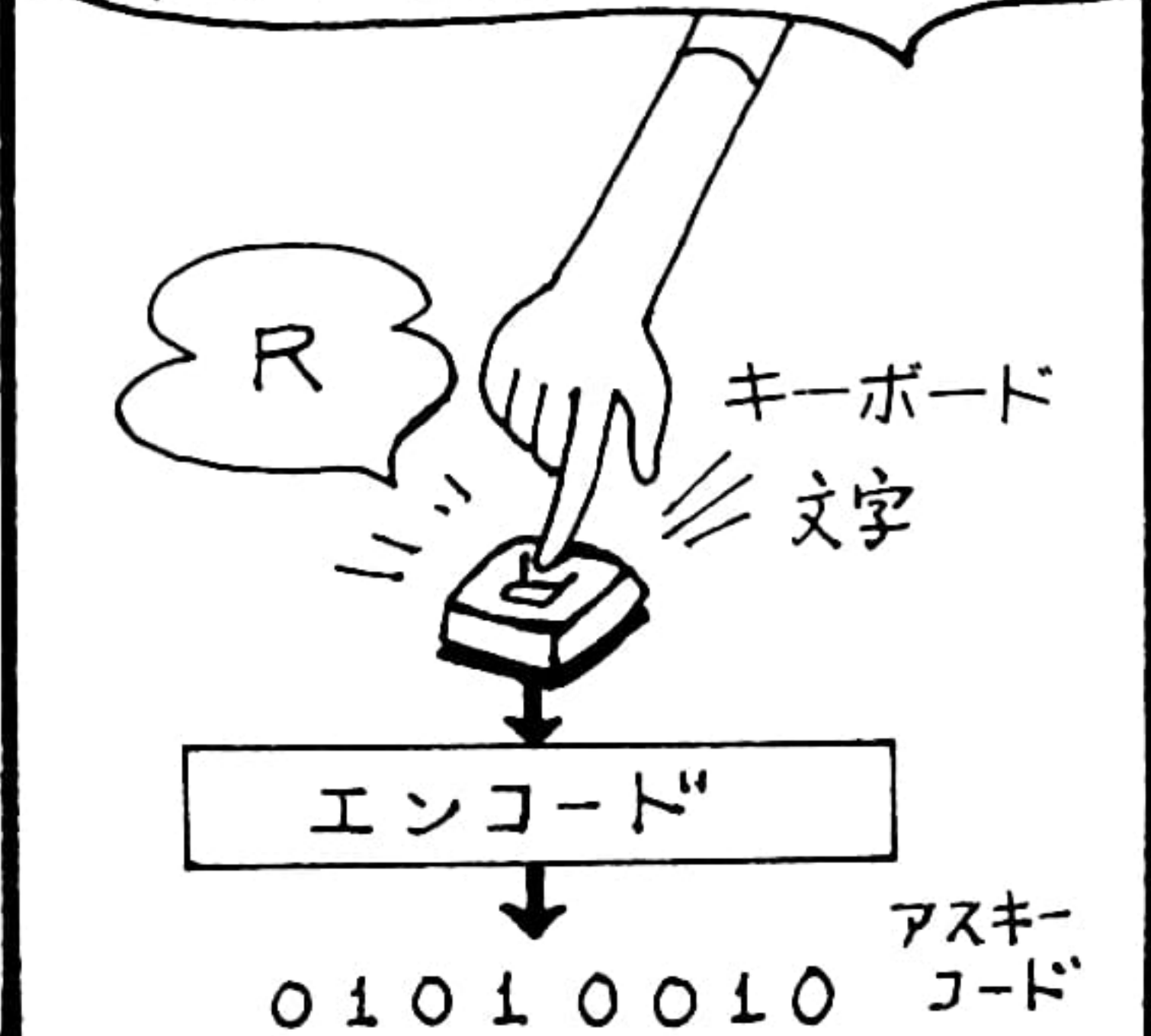
0101 0010

このデータがマシンコードなのか数値なのかアスキーコードなのかはプログラムしだい。

メモリ　アドレス

インプットの時はこの文字記号をエンコードしてアスキーコードにします。

R

キーボード　文字

エンコード

01010010　アスキーコード

そこで説明書などには16進表現で書いてあります。

Rのコードは52です。

0 1 …… 5 …… F

0 1 2 …… F

R

人間同士でアスキーコードの話をする時や説明をしたい時にはコードそのものでは非常に不便です。

01010010

?

このようにアスキーコードというのは7ビットで表現されるコードに文字や記号を割り当てたものですが、

00 1Fまでの32通りのコードには文字記号が割り当ててありません。ここには制御用コードが必要に応じて割り当てられるようになっています。

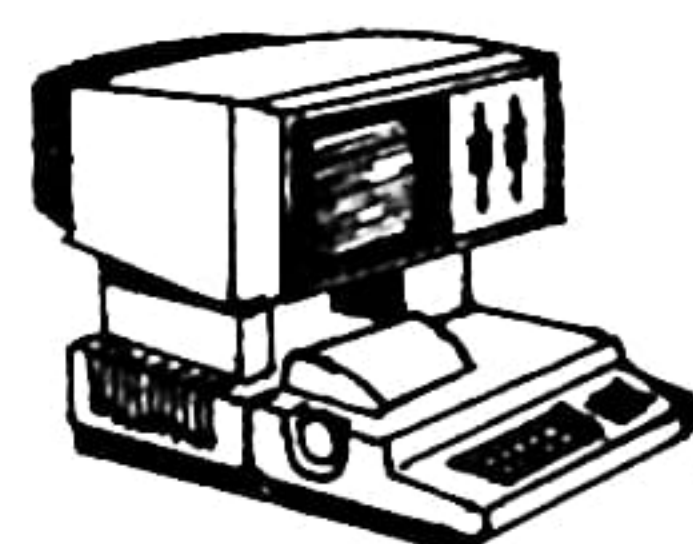
	0	1
0	文字・記号は	割り当てられていない
1		
2		
3		
4		
5		

何番目に割り当てられた文字かを知りたい時は10進表現の方がよくわかりますが、実用性はどうでしょう。

&H52は0から数えて83番目、数字では82ということになります。

BASICでは16進表現であることを示すために&Hをつけます。ポケコンは&だけでした。

&52

&H52

たとえば10進数の5は35というコードになり、このコードを10進数で表現すると53になります。

どうしてこれが同じなの？

5

53 (アスキー10進)

数字は30から39に0から9が割り当てられています。

	3
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	:

次の20から文字記号が割り当てられています。20は空白コードです。20をデコードしたら␣(空白)になります。

	2
0	空白
1	!
2	"
3	#
4	$
5	%
6	&

アスキーコードで文字記号を見た場合、数字＜英大文字＜英小文字の順になっています。

上桁 3ビット (0〜7)

下桁 4ビット (0〜F)

0	1	2	3	4	5	6	7
制御コード		特殊記号	数字	英大文字	記号	英小文字	記号

英大文字は41のAからはじまって5AのZまで、英小文字は61のaから7Aのzという具合になっています。

	4	5	6	7
0	@	P	'	p
1	A	Q	a	q
2	B	R	b	r
3	C	S	c	s
4	D	T	d	t
5	E	U	e	u
6	F	V	f	v
7	G	W	g	w

③BASICでアスキーコードを操作

ASC()
CHR$()

ポケコンの場合はCHR$とASCだけでした。

A$=CHR$(I)
R
I=ASC(A$)
I=VAL("&H"+B$)
82
52
B$=HEX$(I)
A$
I
B$

それとは反対に文字記号を10進表現の数値に変換する関数がASC関数です。

I=ASC(A$)
R
82
A$
I

アスキーと発音します。答が10進なので表を見る時は不便です。

CHR$は10進数値をキャラクタコードとみなしてそれに対応する文字記号(キャラクタ)を与えるものです。

キャラクタダラーと発音します。

R
A$=CHR$(I)
82
A$
I

アスキーコードを10進で表わしています。

16進の文字型コードからは直接変換できる関数が見当たりません。そこで、16進であることを示す&HをつけてVAL関数で10進表現の数値に戻しました。

I=VAL("&H"+B$)
82
52
BASIC
I

バリューと読みます。文字型の数値を数値に戻します。

HEX$は10進数値を16進数表現に変換する関数です。変換された数値は"文字"になります。

52
B$=HEX$(I)
82
B$
I

ヘキサダラーと呼びます。

一番困ったのはHEX$の逆関数が見当たらないということでした。ふと浮かんだのがVAL関数でした。頭に&Hをつけたらこれを16進と解読してくれないでしょうか?

35
42
6
23
451
38
ブラックホール
HEX$( )

どういうわけか、文字記号と16進コード間の変換関数はありません。コード変換表には文字記号とアスキーコードを対応させていますが10進では書いてありません。このへんがややこしいところです。

BASIC
R
A$
52
B$

マイコン同士の通信
RS-232C シリアルデータ転送
アスキーコードと関数
現在位置
文字列と関数
RS-232CとBASIC
サンプルプログラム

話がちょっとそれますが、これも通信に必要な基礎知識です。

ここで大事なのは、なぜ'&H'をプラスすることに気がついたのかということです。そこで文字列についてもう少しくわしくお話しておきましょう。

次にA$='ABC'は、文字列ABCをA$に入れるということです。

ABC
A$='ABC'
A$

まずA=ABCで実行される内容を図示するとこのようになります。AにABCの内容を移すものです。

A=ABC
A
ABC

ここに書いた2つの式に数値と文字列の違いを見ることができます。

A= ABC …①
A$= "ABC" …②
①と②は中身が違う

まず文字と文字をつなぐことができます。たし算の形で結果が得られます。

A$="ABC"+B$+"GHI"
この時 B$の内容が"DEF"とするとこの式の実行後 A$は
ABCDEFGHI
A$
となります。

文字列では演算の代わりに編集ができるようになっています。もちろんBASICレベルです。

A
A$

数値変数か文字変数かは名前の最後に$マークがあるかないかで決まります。

文字型です。
A$

RIGHT$は文字列右側からN個取り出す関数です。

RSTUVWXYZ
B$
A$=RIGHT$(B$,4)
WXYZ
A$
ライトダラー

LEFT$は文字の左側からN個取り出す関数です。

ABCDEFGH
B$
A$=LEFT$(B$,3)
ABC
A$
レフトダラー

その他文字を分けたり、数えたり、数値化、文字化したりする関数があります。

LEFT$
RIGHT$
MID$
STR$
VAL
LEN

LENは文字列の長さを示す関数です。結果は数値になります。

ABCDEFGHI

B$

A = LEN(B$)

9

A

レングス

MID$関数はランダムファイルで重要な役割を果たしました。

MID$は文字列の左側から数えてI番目の文字からN個取り出す関数です。

1 2 3

ABCDEFGHIJKLMN

B$

A$ = MID$(B$, 3, 4)

CDEF

A$

ミッドダラー

IF800-30の場合は'&H'をつけると16進として0～9、A～Fを受け付けてくれました。

46

A$

A = VAL("&H"+A$)

71

A

46は文字、71は数値です。

VALはSTR$の逆関数です。0～9のキャラクタから成る文字列を数値に変換する関数です。

3.14G578

A$

A = VAL (A$)

3.14

A

バリュー

STR$は数値を文字列に変換する関数です。ランダムファイルで使ったMKS$関数とはタイプが違います。

12.34

A

A$ = STR$(A)

12.34

A$

ストリングダラー

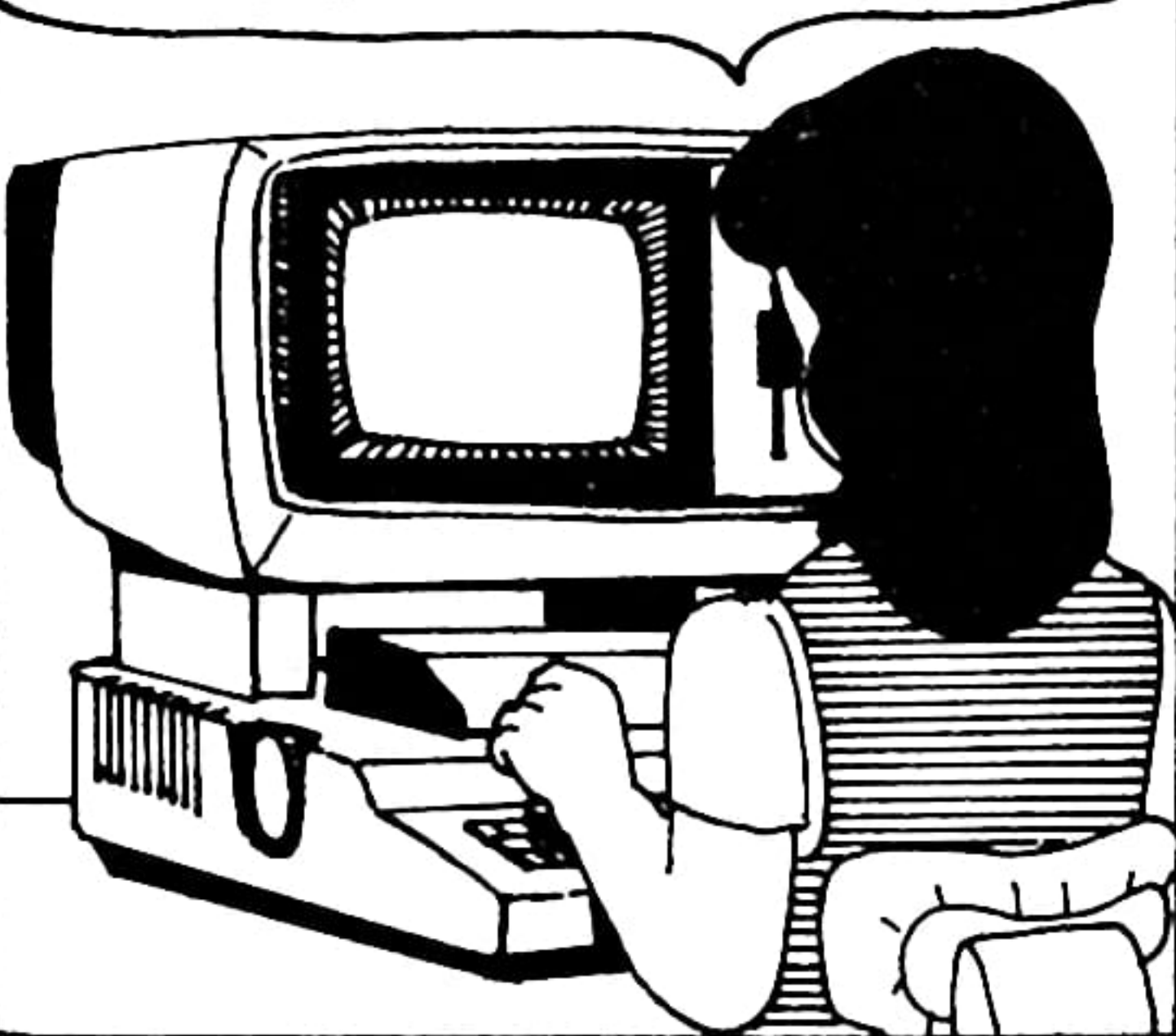
この秘密は文字列の各キャラクターが計算機内部ではアスキーコードになっていることにあります。

アスキーコードの数値表現	49	97	65
アスキーコード&H	31	61	41
キャラクタコード	1	a	A

また、いくつかの文字列をキャラクタ順に並べることができます。

いずれがあやめかかきつばた

1 a A

では、まとめの代わりにサンプルプログラムでこのうちのいくつかの関数を確かめてみます。

論理IF文を使うと文字列を思いのままに並べ換えることができるわけです。

IF␣A$ > B$␣THEN

ETUKO

A$

TOMOKO

B$

1、a、Aの3つのキャラクタはコード化すると、10進数値ではそれぞれ49、97、65となりこれを大きい順に並べると97、65、49です。元のキャラクタに戻すとa、A、1となるでしょう。

a

A

1

これは画面のハードコピーです。RUNすると=が表示されました。アスキーコード3Dはキャラクタで=になっていますから&H3DはOKです。

```
Ok
AUTO                  RETURN
10 A$=CHR$(&H3D)      RETURN
20 PRINT A$           RETURN
30 END                RETURN
40                    CAN
Ok
RUN                   RETURN
=
Ok
```

まず16進コード3Dに&Hをつけて A$=CHR$(&H3D)がどうなるか、みてみます。

プログラムをキーインする。

```
10␣A$=CHR$(&H3D)
20␣PRINT␣A$
30␣END
```

このアスキーコードに"&H"をつけてキャラクタコードに換えるテクニックは通信の時かなり応用できるはずなんですが…。

VAL

"&H"+"3D"は"&H3D"という文字列になってしまったのです。CHR$( )( )の中は文字列を使うと文法違反です。

さあ困った

これに気を良くして、行番号10をA$=CHR$("&H"+"3D")としてRUNしてみました。

```
EDIT␣10
10␣A$=CHR$("&H"+"3D")
RUN
Type mismatch in 10
OK
```

ピッ

すると、タイプ・ミスマッチとなってエラーでした。

```
10 INPUT "16ｼﾝ ASCII ｺｰﾄﾞ ﾃﾞｰﾀ 2ｹﾀ =";A$
20 B$=CHR$(VAL("&H"+A$))
30 PRINT A$;"...";B$:LPRINT A$;"...";B$:PRINT :LPRINT
40 GOTO 10
```

```
38...8
39...9
3B...;
3D...=
3F...?
```

サンプルプリントです。

アスキーコードを16進2桁で入れてそのコードに対応するキャラクタを表示するプログラムです。

ここで浮んだのがVAL関数です。VAL( )の( )の中は文字列だからひょっとしたら…。

```
EDIT␣10
10␣A$=CHR$(VAL("&H"+"3D"))
RUN
=
OK
```

うん、OKです。

アスキーコードと文字列の予備知識はこれぐらいにして、いよいよRS-232Cにチャレンジです。なお次のページには10進数値併記のコード表を載せていますから参考にしてください。

これも画面のハードコピーです。プリントではありません。

```
Ok
LIST
10 INPUT A$
20 B=VAL ("&H"+A$)
30 C$=CHR$ (B)
40 C=ASC(A$)
50 PRINT A$,B,C$,C
Ok
RUM
RUN
? 45
45            69            E            52
Ok
```

アスキーコードをインプットし、まず10進数値を求め、次にキャラクタを求めるものです。行番40では"45"の"4"が10進数値に変換されて52となっています。

キャラクタ・コード表

ここに載せているキャラクタはASCIIやJISコード指定席です。余白のコードにはメーカーが独自のキャラクタを割り付けていますので、お手持ちのパソコンマニュアルのキャラクタ・コードを見てください。キャラクタAの&H(16進)表現は上桁(4・)と下桁(・1)をつないで&H41となります。&H41=65Dです。

上桁 / 下桁	0・		1・		2・		3・		4・		5・		6・		7・		8・		9・		A・		B・		C・		D・		E・		F・	
	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ	10進数値	キャラクタ
・0	0		16		32	空白	48	0	64	@	80	P	96		112	p	128		144		160		176	ー	192	タ	208	ミ	224		240	
・1	1		17		33	!	49	1	65	A	81	Q	97	a	113	q	129		145		161	。	177	ア	193	チ	209	ム	225		241	
・2	2		18		34	"	50	2	66	B	82	R	98	b	114	r	130		146		162	「	178	イ	194	ツ	210	メ	226		242	
・3	3		19		35	#	51	3	67	C	83	S	99	c	115	s	131		147		163	」	179	ウ	195	テ	211	モ	227		243	
・4	4		20		36	$	52	4	68	D	84	T	100	d	116	t	132		148		164	、	180	エ	196	ト	212	ヤ	228		244	
・5	5		21		37	%	53	5	69	E	85	U	101	e	117	u	133		149		165	・	181	オ	197	ナ	213	ユ	229		245	
・6	6		22		38	&	54	6	70	F	86	V	102	f	118	v	134		150		166	ヲ	182	カ	198	ニ	214	ヨ	230		246	
・7	7		23		39		55	7	71	G	87	W	103	g	119	w	135		151		167	ァ	183	キ	199	ヌ	215	ラ	231		247	
・8	8		24		40	(	56	8	72	H	88	X	104	h	120	x	136		152		168	ィ	184	ク	200	ネ	216	リ	232		248	
・9	9		25		41	)	57	9	73	I	89	Y	105	i	121	y	137		153		169	ゥ	185	ケ	201	ノ	217	ル	233		249	
・A	10		26		42	*	58	:	74	J	90	Z	106	j	122	z	138		154		170	ェ	186	コ	202	ハ	218	レ	234		250	
・B	11		27		43	+	59	;	75	K	91		107	k	123		139		155		171	ォ	187	サ	203	ヒ	219	ロ	235		251	
・C	12		28		44	,	60	<	76	L	92		108	|	124		140		156		172	ャ	188	シ	204	フ	220	ワ	236		252	
・D	13		29		45	−	61	=	77	M	93		109	m	125		141		157		173	ュ	189	ス	205	ヘ	221	ン	237		253	
・E	14		30		46	.	62	>	78	N	94		110	n	126		142		158		174	ョ	190	セ	206	ホ	222	゛	238		254	
・F	15		31		47	/	63	?	79	O	95		111	o	127		143		159		175	ッ	191	ソ	207	マ	223	゜	239		255	

10進数値と併記しているのでASC関数が使いやすくなります。

④まずはハードのつなぎ方

まず、ハード的に何が必要かをみます。 コネクタとケーブルが必要なのは当然ですが、ポケコンなどRS-232Cポートがついていないものの場合RS-232Cインターフェイスユニットが必要です。

インターフェース
CE 158

コネクタは25ピン、サイズや信号の電圧レベルなどが規定されていますが、どのピンにどの信号が出ているのかはメーカーやユニットによってまちまちなのが現状です。

そこでコネクタに結線する時は各ユニットの取扱説明書のピン番号と信号の関係をよく見ることが必要です。

でも使っているピンは25ピンのうちせいぜい8ピンです。信号の持っている意味さえ理解しておけばすぐ対応できます。

13 1
25 14

グランド線を除けばあと7つ。これらは出力か入力かのいずれかで、ピンとユニットへの回路の入口は下図のようになっています。

出力ピン
グランド
入力ピン

こちらの出力ピンは相手の入力ピンに、相手の出力ピンはこちらの入力ピンに接続します。

出力ピン 入力ピン
グランド

データ用のピンは送信用、受信用それぞれ1個ずつあります。こちらの送信側は相手の受信側に接続します。

送 受 送 受

あと2組の受信状態を確認するための信号と、キャリア検出という信号があります。キャリアはモデムを使う時にだけ関係があります。

送信要求 …(出) 送信してくれと要求する。
送信可 …(入)
受信要求 …(出) 受信してくれと要求する。
受信可 …(入)
キャリア検出…

送信要求はINPUTのような受信文が実行された時、受信要求はPRINTのような送信文が実行された時それぞれ出ていくと考えられます。

プログラムでは双方同時に送信要求や受信要求が出るとは考えられません。片方がデータを送るタイミングではもう一方はデータを受信するようにプログラムされているはずです。

PRINT#1
INPUT A

でもこんなことは結線時にちょっと注意しておけばよいだけです。BASICレベルのプログラムではまったく意識する必要がありません。

PC1500側

			ピン
送信データ	… TD	出力	②
受信データ	… RD	入力	③
送信要求	… RTS	出力	④
送信可	… CTS	入力	⑤
データセットレディ	… DSR	入力	⑥
信号用接地	SG		⑦
キャリア検出	… CD	入力	⑧
端末装置レディ	… DTR	出力	⑳

2つのマニュアルの中の信号名をそのまま転記しました。

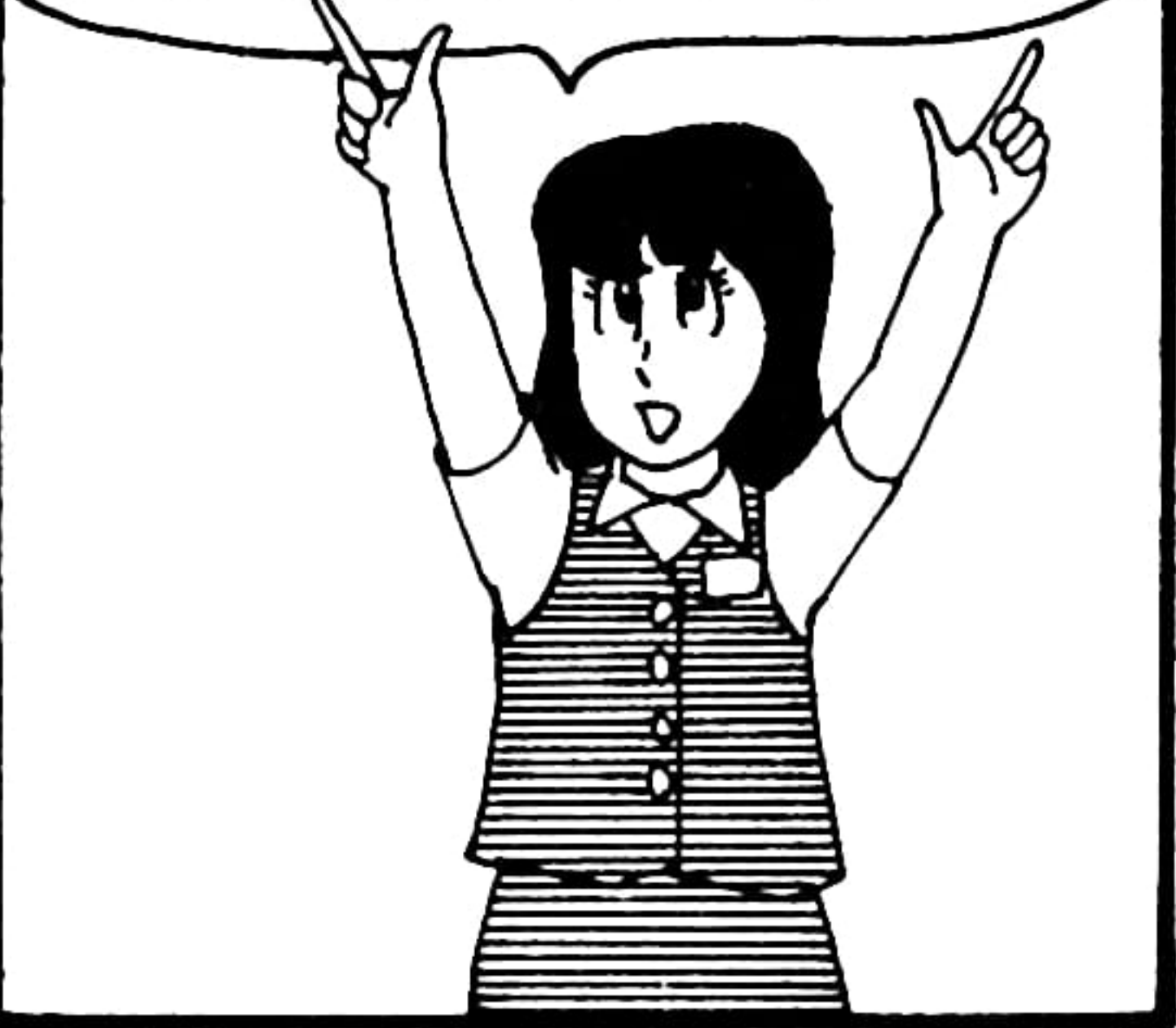
IF800-30側

ピン			
②	送信データ	… BA	入力
③	受信データ	… BB	出力
④	送信要求	… CA	入力
⑤	送信可	… CB	出力
⑥	受信要求	… CC	出力
⑦	信号グランド		
⑧	キャリア検出	… CF	出力
⑳	受信可	… CD	入力

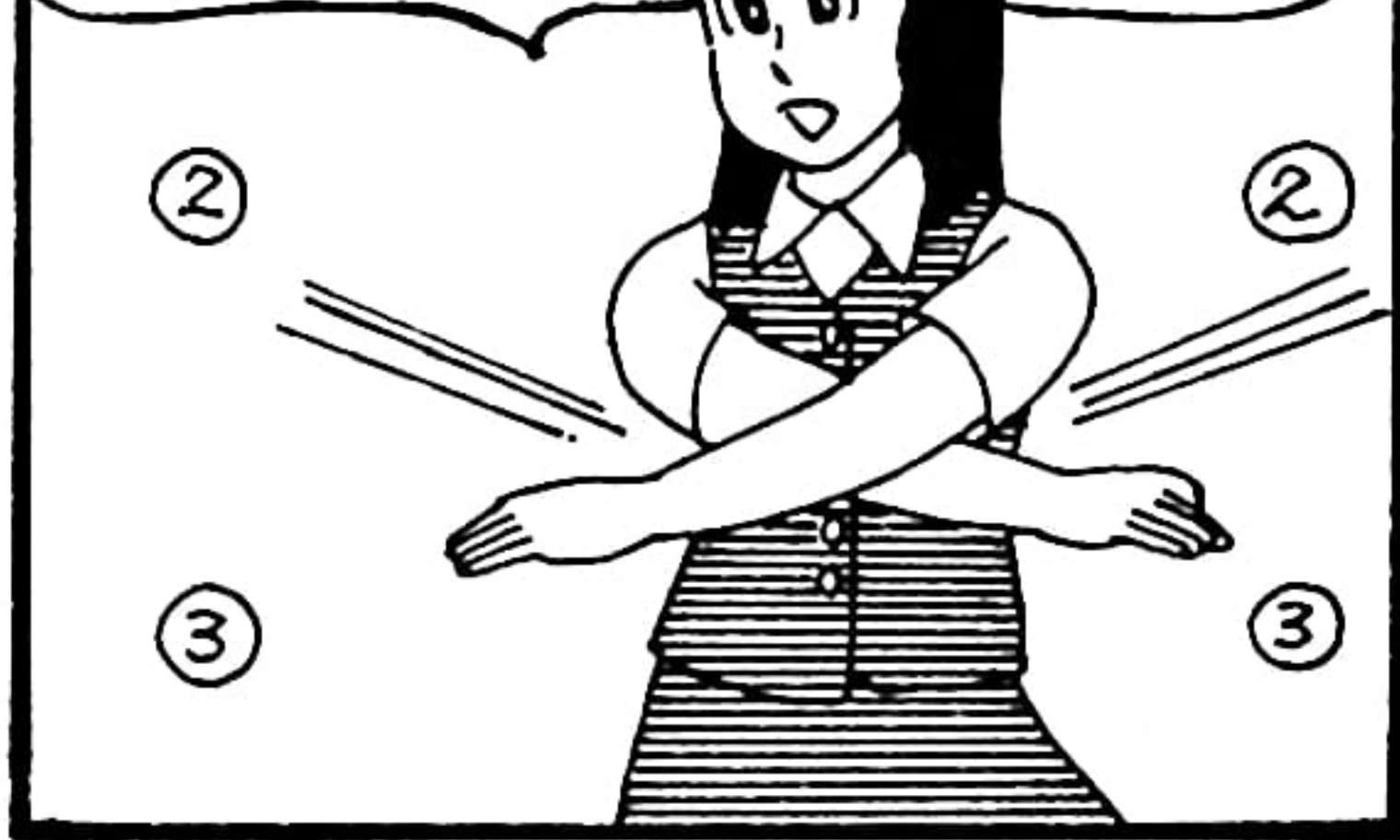
ところが、両方ともBUSYとWAITの状態になってどうしてもうまくいかないんです。

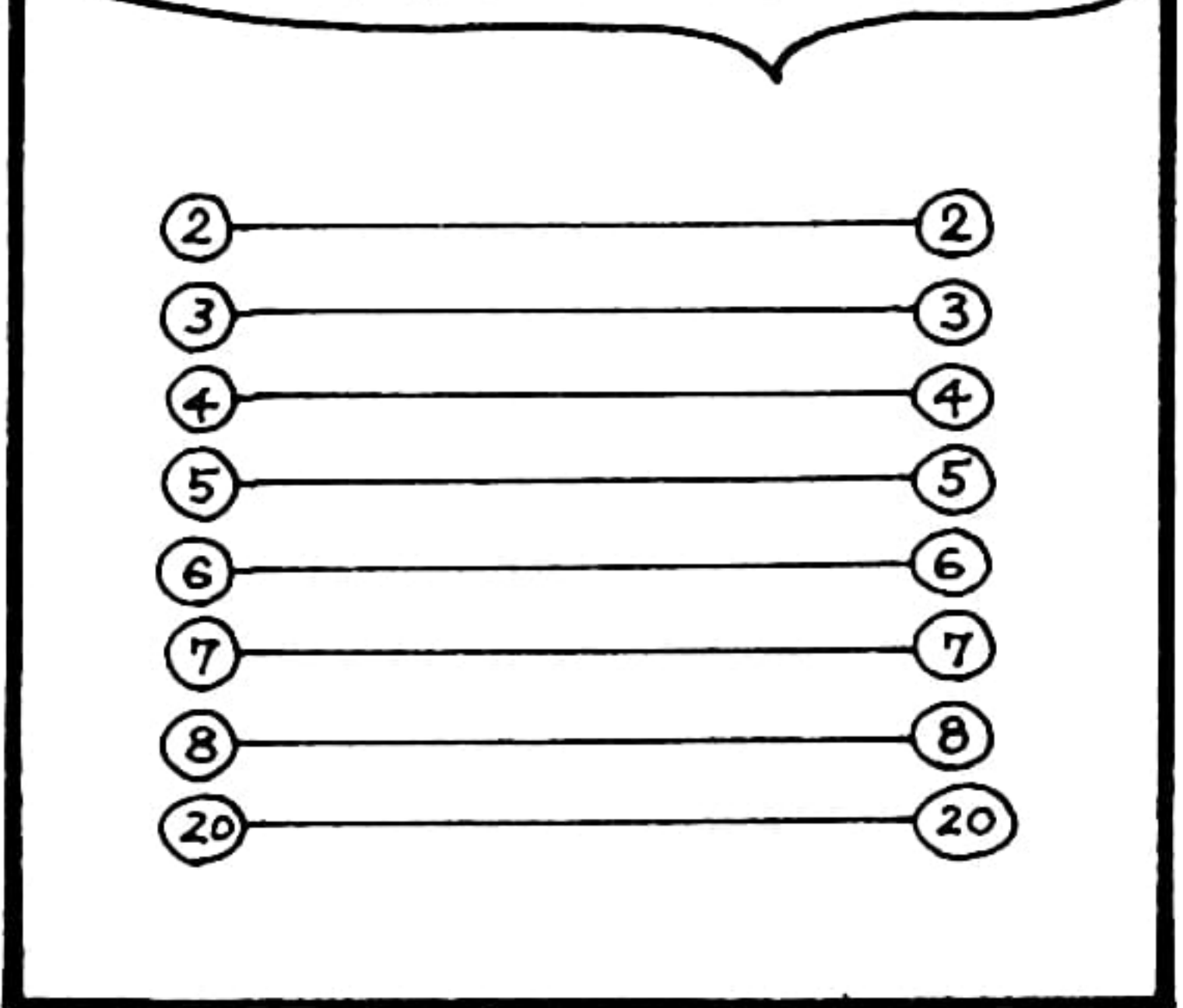
………。

……。

上の2つの信号名から考えても、この場合クロスケーブルが必要なことは不思議ではありません。

同機種をつなぐ時は、一方の2番ピンと、もう一方の3番ピンを対にするようなことになります。このように接続された対のコネクタを持つケーブルをクロス・ケーブルと言っています。

② ②

③ ③

なんのことはない。この組み合せでは、クロスせず、まっすぐにつなげばよかったのです。

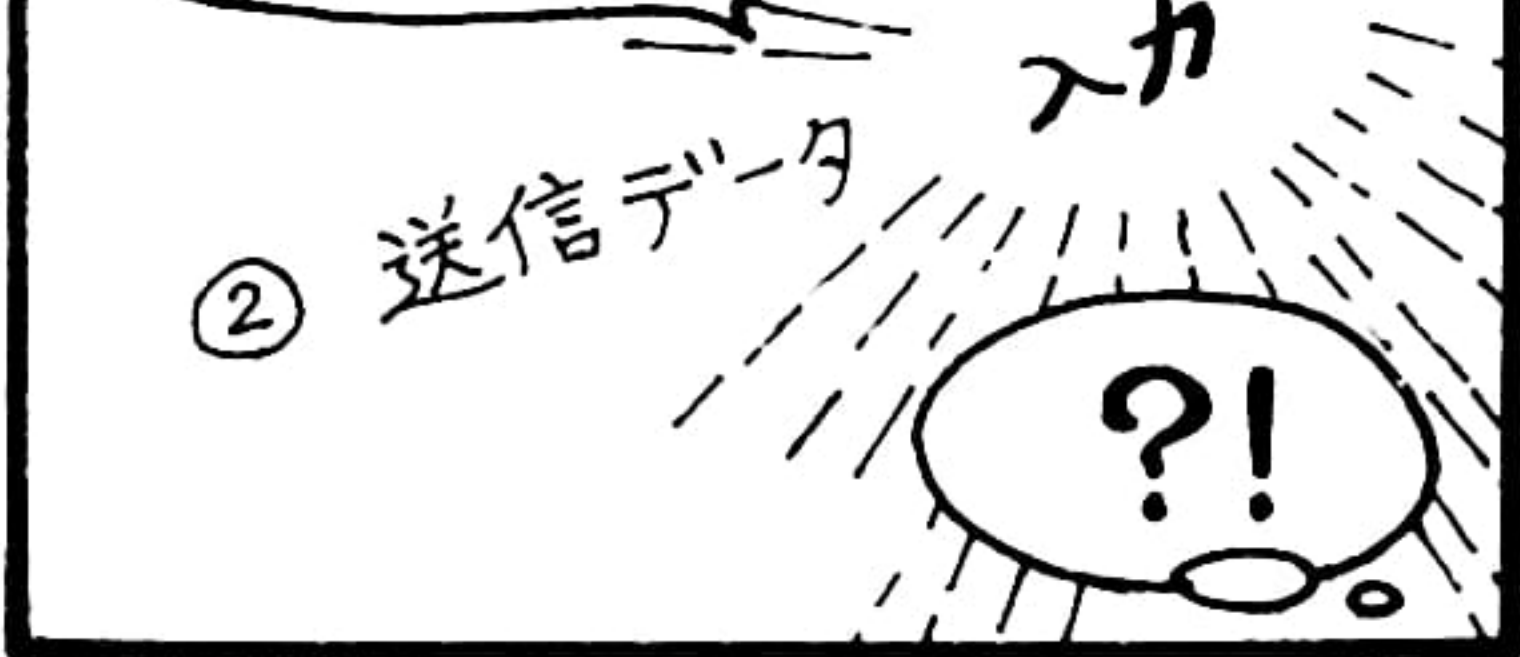
②—②
③—③
④—④
⑤—⑤
⑥—⑥
⑦—⑦
⑧—⑧
⑳—⑳

わっ、わっ、全部さかさまじゃないの。

IF800とPC1500では、「入力」と「出力」の表示が反対だったのです。

IF800 30

ところがある日、IFのマニュアルをなんとなく見ていた時、入出力の方向がポケコンと反対なのはどういうことだろうと、もう一度考えてみたんです。

② 送信データ 入力

?!

まず用意したプログラムをインプットしながらそれぞれの文の意味を説明することにしましょう。

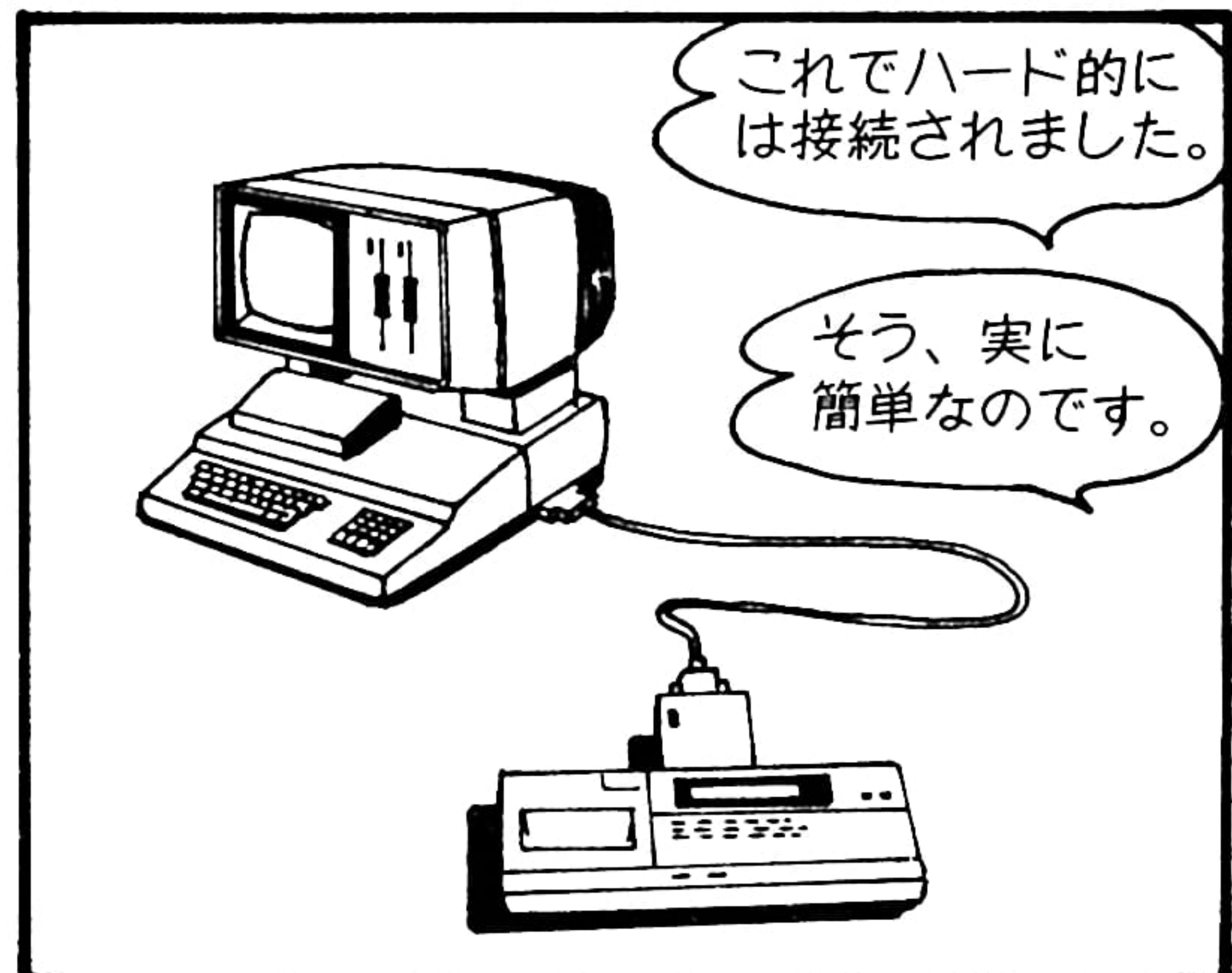
まずIFから

カチャカチャ

では、まず送受信のテストプログラムをRUNさせてみましょう。

SHARP CE158

これでハード的には接続されました。

そう、実に簡単なのです。

⑤RS-232C活用のソフト

```
AUTO
10 OPEN "COM1:3N71" AS 1
```

まず通信条件の宣言です。ここはパソコンによってそれぞれ書き方が違います。

ボーレートなど通信のための条件はプログラムでセットします。

3 N 7 1 " AS 1

通信速度 300ボー

ストップビット数

バッファの番号

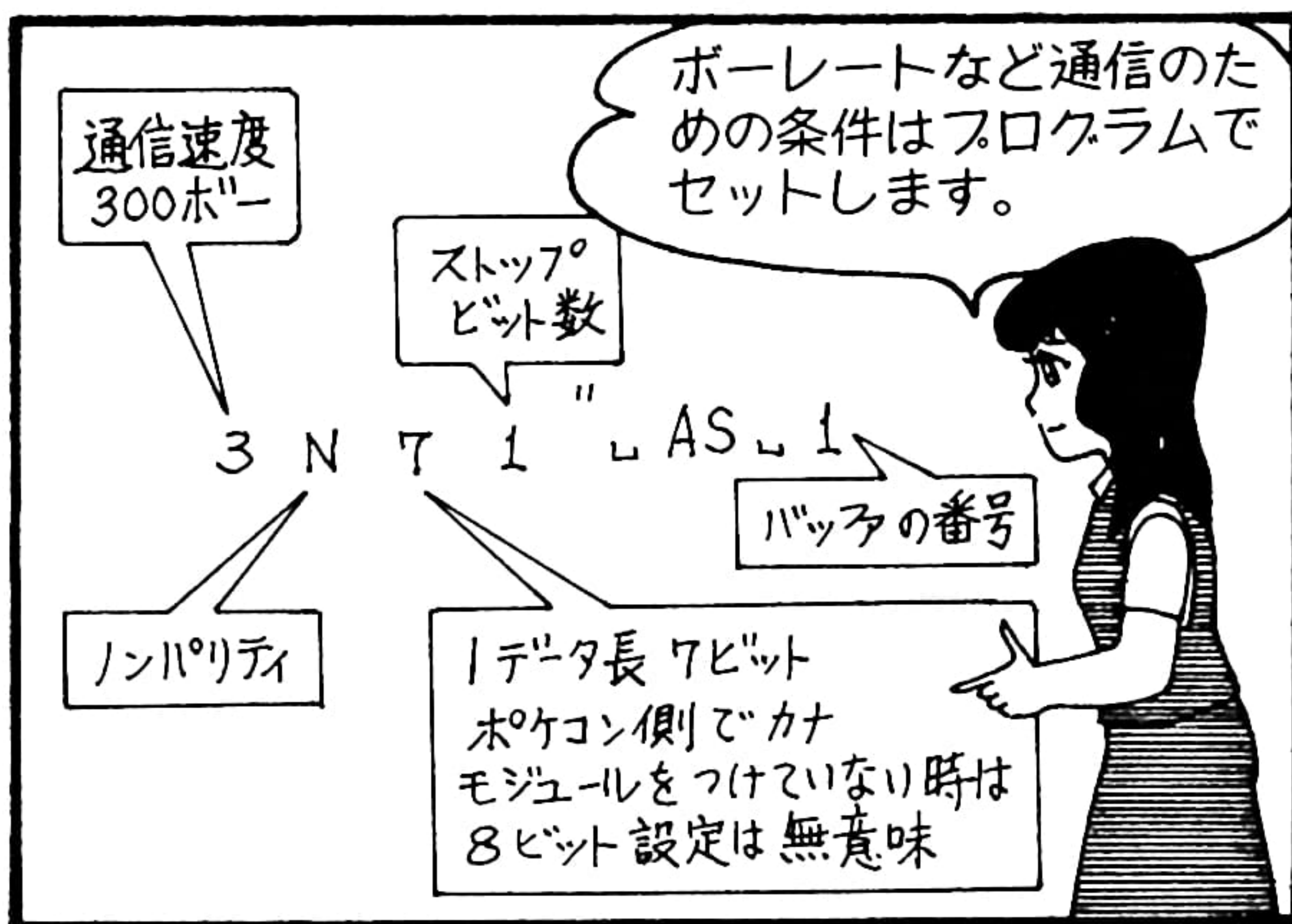
ノンパリティ

1データ長 7ビット ポケコン側でカナモジュールをつけていない時は8ビット設定は無意味

文30はバッファレジスタ#1から変数F$にデータを取り出す命令です。

```
20 PRINT "RS-232C TEST PROGRAM"
30 INPUT #1,F$
```

文40は変数F$の内容を表示する命令です。これでポケコンから送られたデータの内容を見ることができます。

```
40 PRINT ,F$
```

文50と60でF$の内容を見てプログラムを続行するか終了するかの判断をしています。F$の内容がENDだったら終了です。

```
50 IF F$<>"END" THEN 30
60 PRINT "TEST END"
70 END
```

RUNすると、IF側はデータ待ちとなります。

```
10 OPEN "COM1:3N71" AS 1
20 PRINT "RS-232C TEST PROGRAM"
30 INPUT #1,F$
40 PRINT F$
50 IF F$<>"END" THEN 30
60 PRINT "TEST END"
70 END

RUN [RETURN]
```

次はポケコン側です。命令文はポケコンのマニュアルを参照しながら、プログラムします。

まず、通信条件の設定はセットコミュニケーション文で行ないます。セットコムとでも呼びましょうか。

```
10:SETCOM 300,7,N,1
```

IF側と同条件になるようにセットしなければなりません。

次はセットデバイス文でPOをセットします。これに対応する命令はLPRINT文です。セットデブと読みましょう。

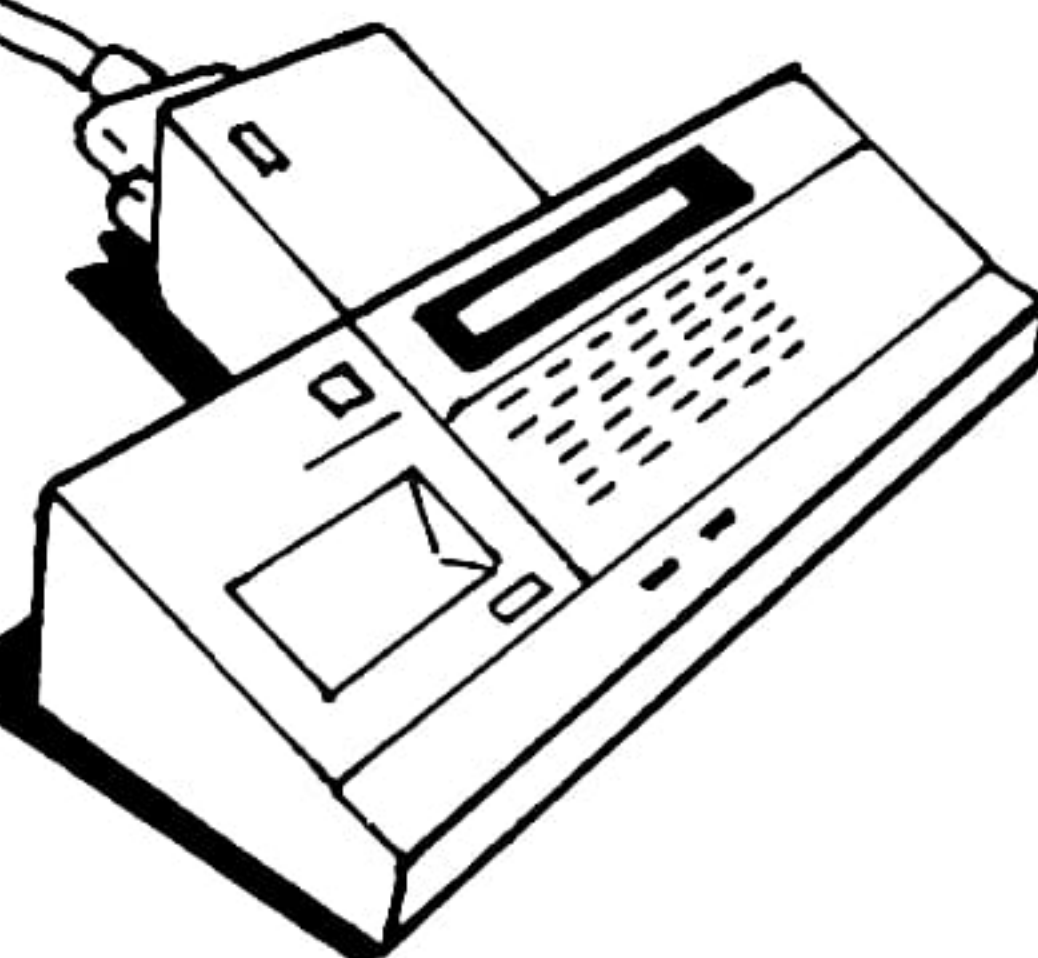
```
20:SETDEV PO
```

解除する時は、SETDEVだけの文にすればよいのです。解除後は印字文となります。

文30は2つの出力RTSとDTRを共にONにする命令です。アウトプットポートのステータスを決める文です。アウトスタットとでも読んでください。

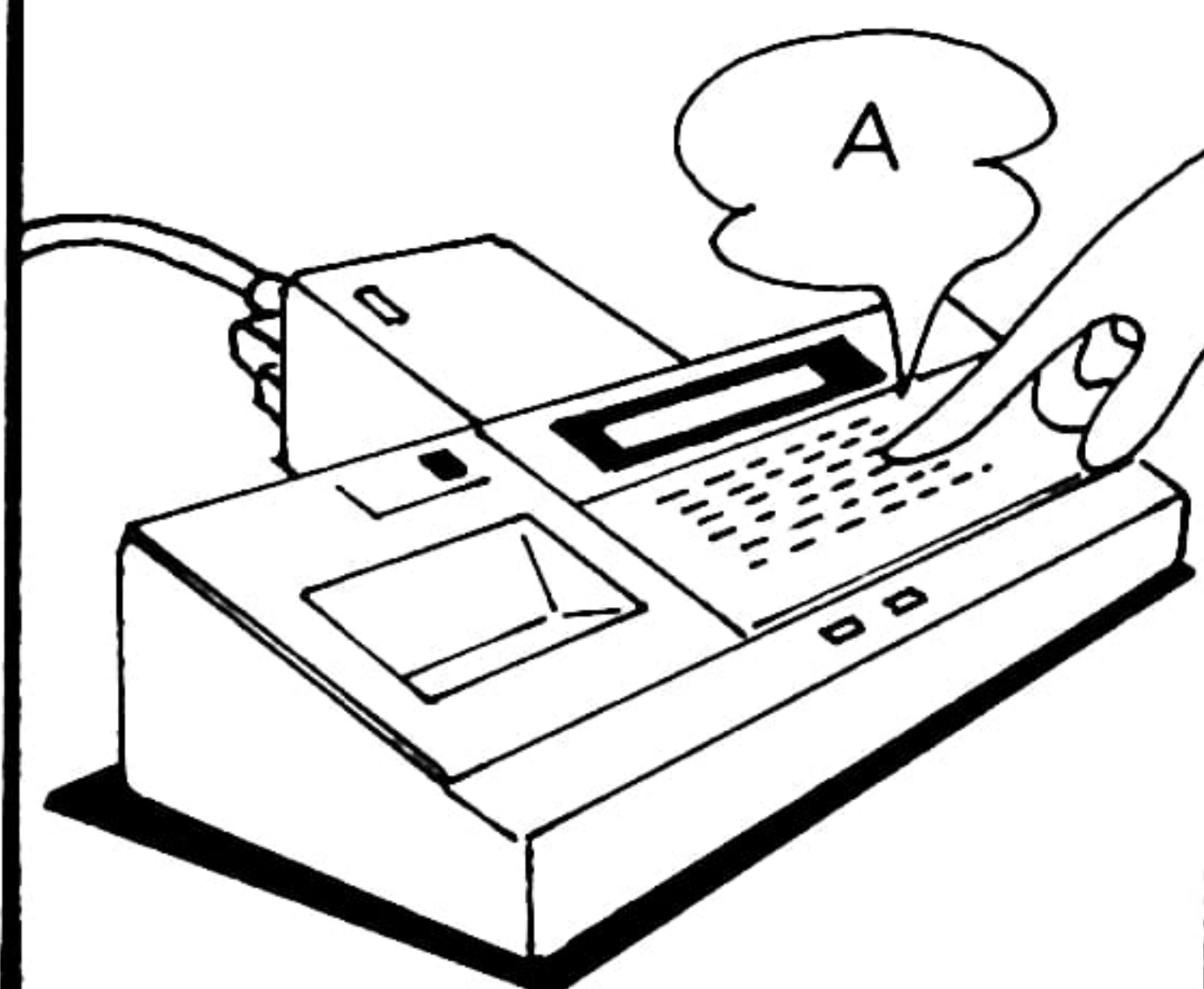
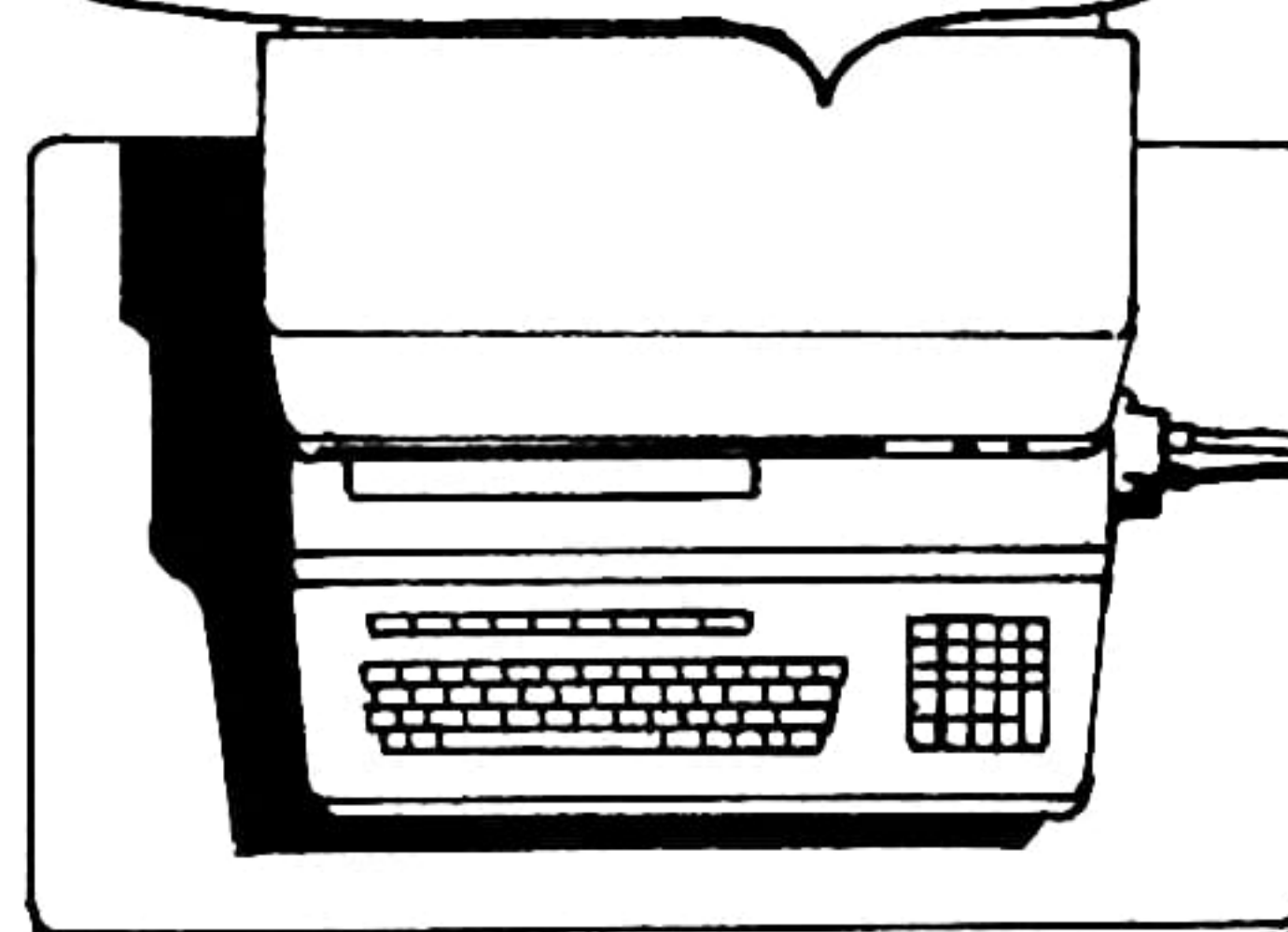
```
30:OUTSTAT 0
```

以下PC-1500からRS-232Cポートに出力する命令を書きます。

```
 5:WAIT200
10:SETCOM␣300,7,N,1
20:SETDEV␣PO
30:OUTSTAT␣0
40:INPUT"DATA=";A$
50:LPRINT␣A$
60:IF␣A$<>"END"␣THEN␣40
70:PRINT"TEST␣END"
80:END
```

ポケコンの場合SETDEV文が実行されると、その後のINPUTやLPRINTなどの命令はRS-232Cポートに対して行なわれます。

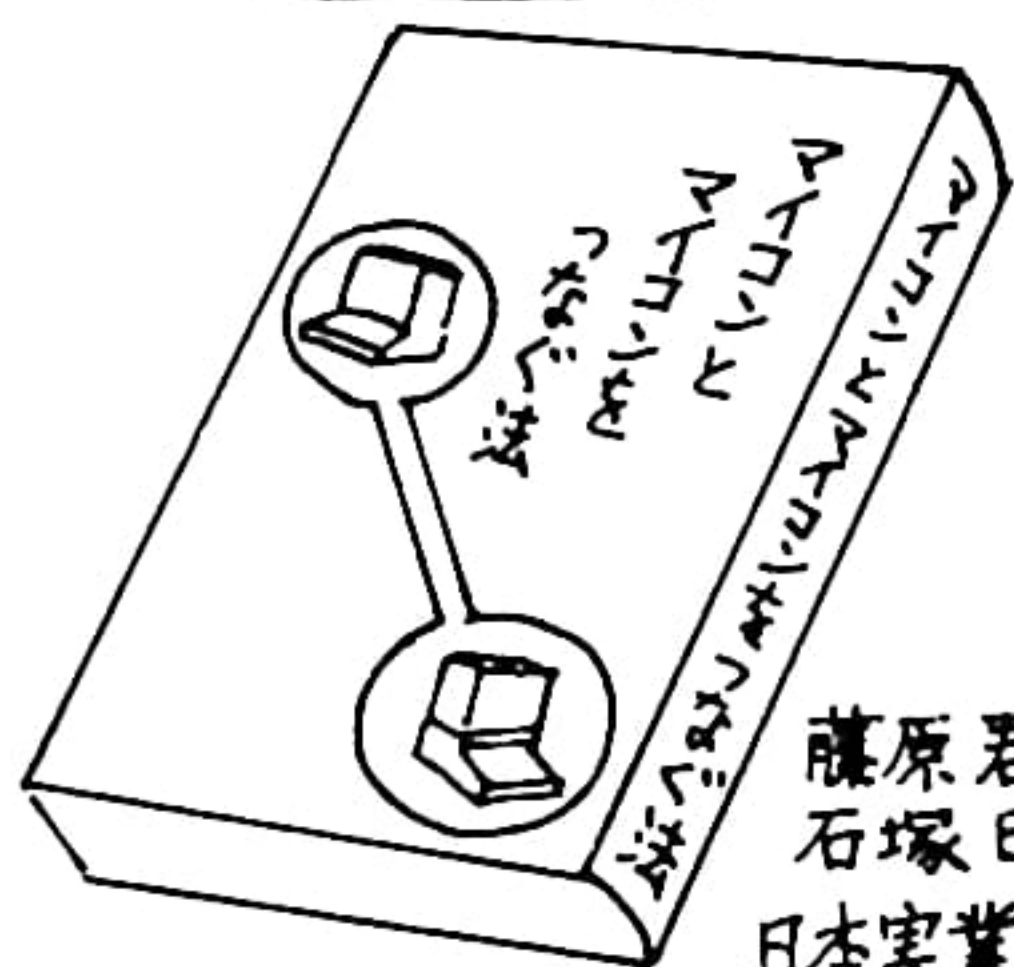
アサインメント	方向	命令文
KI	(INPUT)	INPUT
PO	(OUTPUT)	LPRINT

まずA[ENTER]です。

A

RS-232C TEST PROGRAM

これでポケコンPC-1500とIF800-30はハード、ソフト共に接続完了しました。

DATA=

もっと、プログラムの説明をしてもいいのですが、いい本がすでに出ていますので省略しましょう。

マイコンとマイコンをつなぐ法

藤原君忠
石塚日出子 著
日本実業出版社

次に I␣LOVE␣ETUKO [ENTER]と入れてみました。

RS-232C TEST PROGR
A
I LOVE ETUKO

やりました。これもOKです。

そして、IFの画面を見ると…

RS-232C TEST PROG
A

やった。PC側で入れたAが表示されました。

別に知らなくてもよいことと、知っていれば発想が広がり応用が効くということがあるものです。

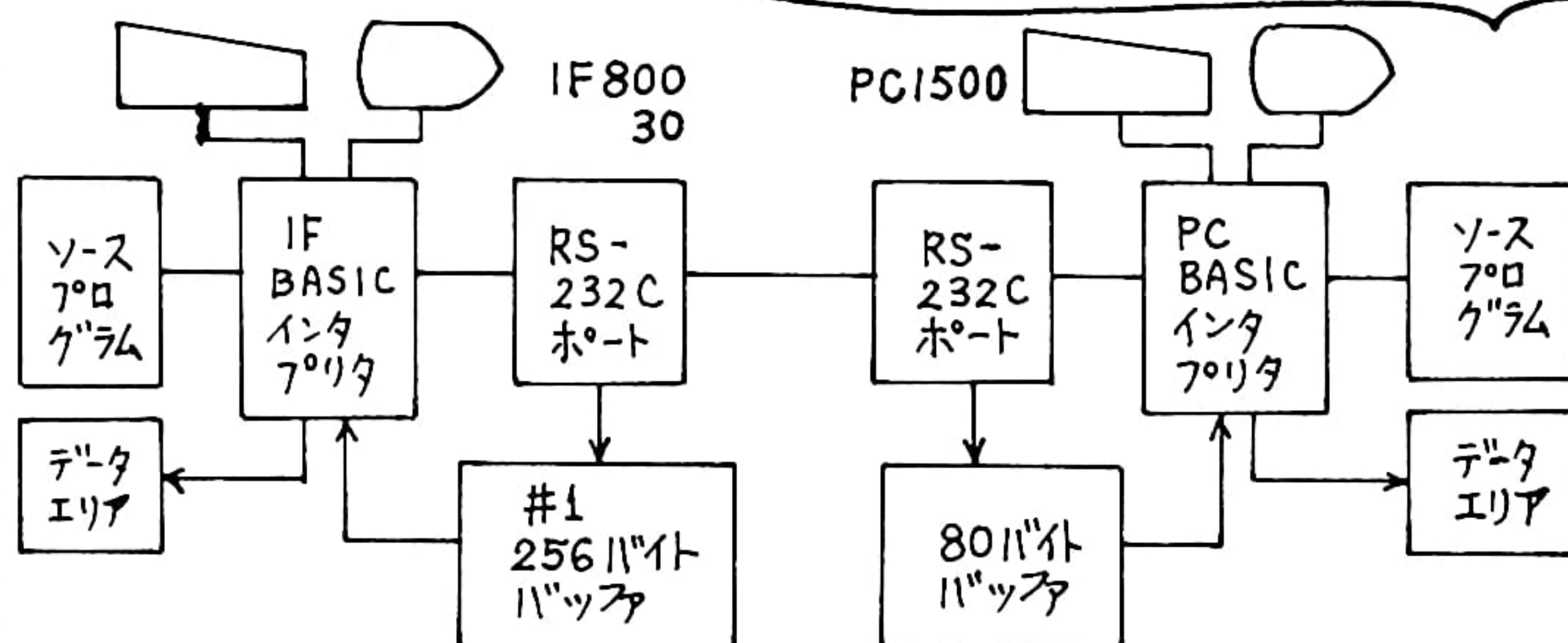
IF800 30

PC1500

ソースプログラム → IF BASIC インタプリタ → RS-232C ポート → #1 256バイトバッファ

データエリア

RS-232C ポート → 80バイトバッファ → PC BASIC インタプリタ ← ソースプログラム

データエリア

そこで、ここではソフトメカニズムについてもう少し考えてみたいのです。

Aはエンコードして&H41つまり01000001とします。
次は、40: INPUT "DATA="; A$だからDATA=を表示してキー入力を待ちます。キー入力されたらそれをA$に入れます。
0100 0001
A
A
BASIC インタプリタ
A$

先にテストプログラムでポケコンから打ち込んだ"A"がどこをどう通ってIFに行って画面に現われたのでしょうか。また、インタプリタちゃんに登場してもらいましょう。
DATA= _

通信条件は10:SETCOM 300,7,N1で宣言済みです。
50:LPRINT A$は、すでにSETDEV␣POが宣言されていますので、A$の内容をRS-232Cポートに出力せよ、ということね。
この意味は速度が300ボー、データ長が7ビット、パリティビットは無、ストップビットは1ということです。
データの区切りとしてラインフィード(LF)コード&HODをつけます。
ポート
RS-232Cのデータ出力ピン
Aのアスキーコード
LFのアスキーコード
100 0001
000 1101
(&H41)
(&HOD)
0100 0001
A$
BASIC インタプリタ

私は30␣INPUT␣#1,F$でデータ待ちだったのよ。
何かデータがきました。10␣OPEN␣"COM1:3N71"␣AS␣1にしたがいバッファ#1にデータを入れます。
IF側
ポート
プリタ
#1
&H41
&HOD
IF 800

IFの方でデータの区切りができたということは、この&HODという制御コードもどうやら機種をこえた指定席のようです。
000 1101
&HOD
100 0001
&H41

ポケコンに打ち込んだ"A"はこのようにしてIFの画面までやってきたのです。
RS-232C TEST PROGRAM
A

次の40␣PRINT␣F$でF$の内容を画面に表示するわ。&H41を表示するとAになります。
A
デコードしてビデオRAMへ
&H41
F$

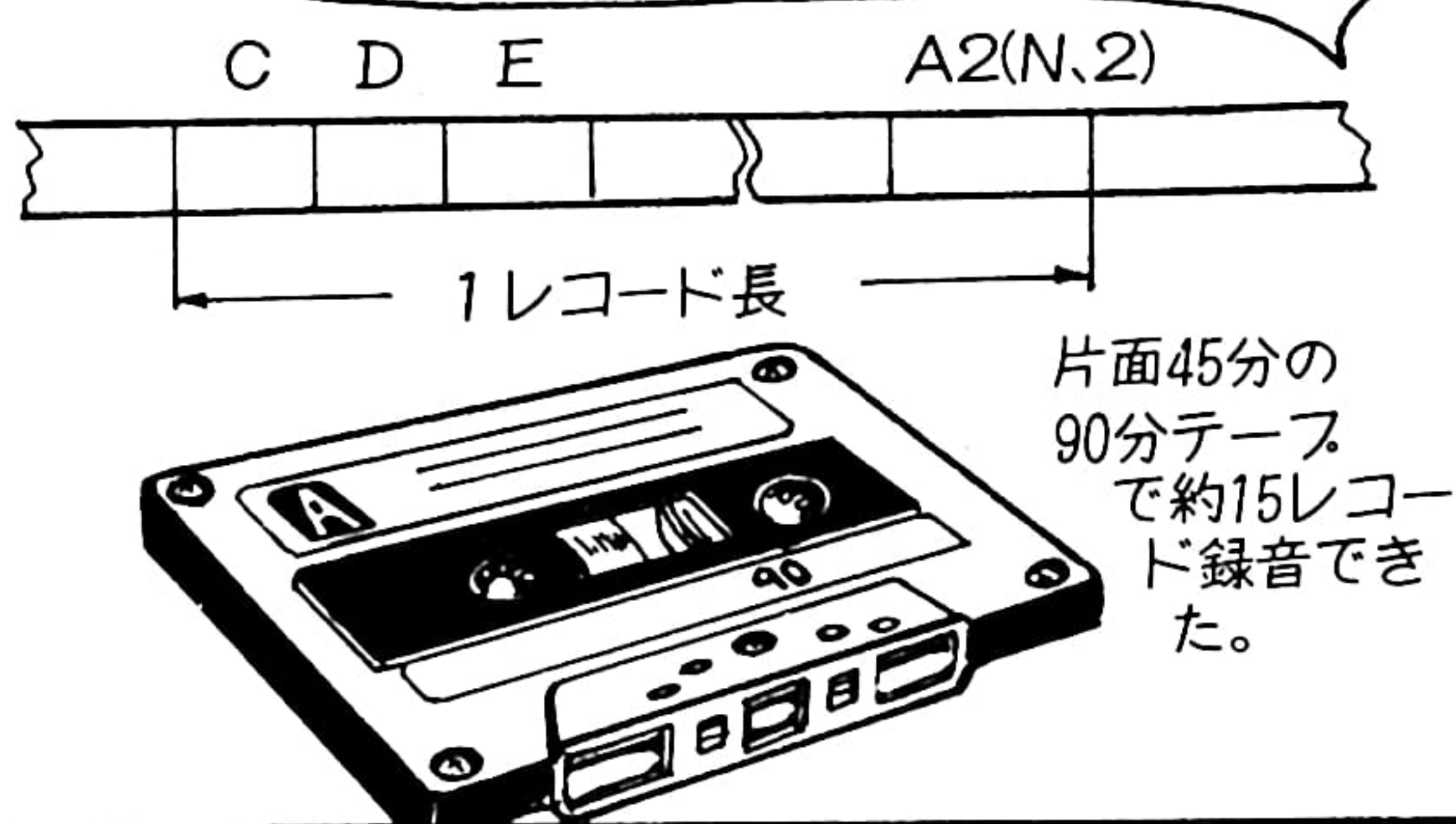
&HODまでのデータをF$に入れるわ。おや、データは&H41が1個だけなのね。
&H41
F$
&H41
&HOD

⑥データの転送

ポケコンやハンドヘルドの場合は価格的な手軽さからカセットにデータのファイルを作ることが多いと思います。この場合はデータが順番に並ぶシーケンシャルファイルになります。

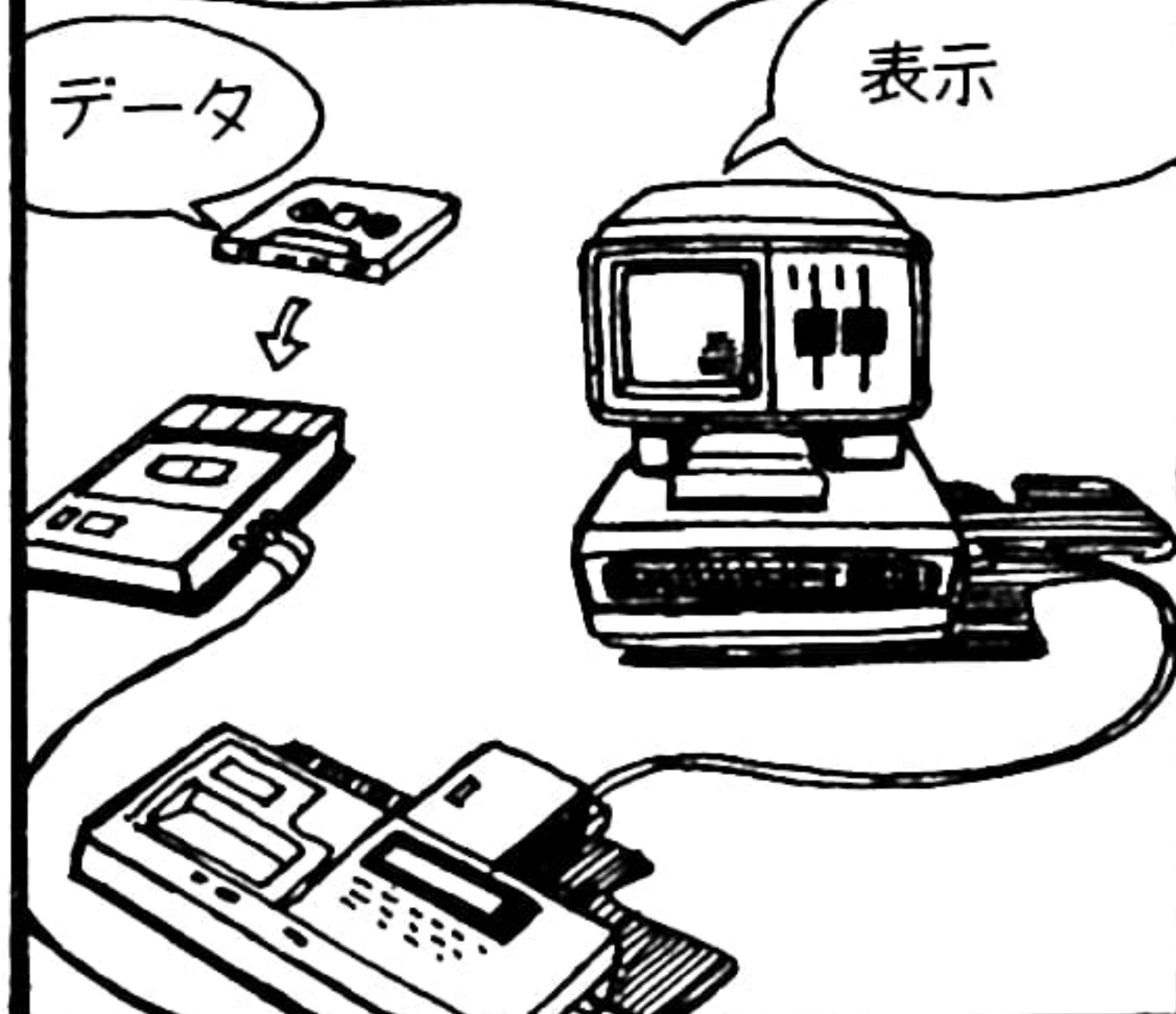
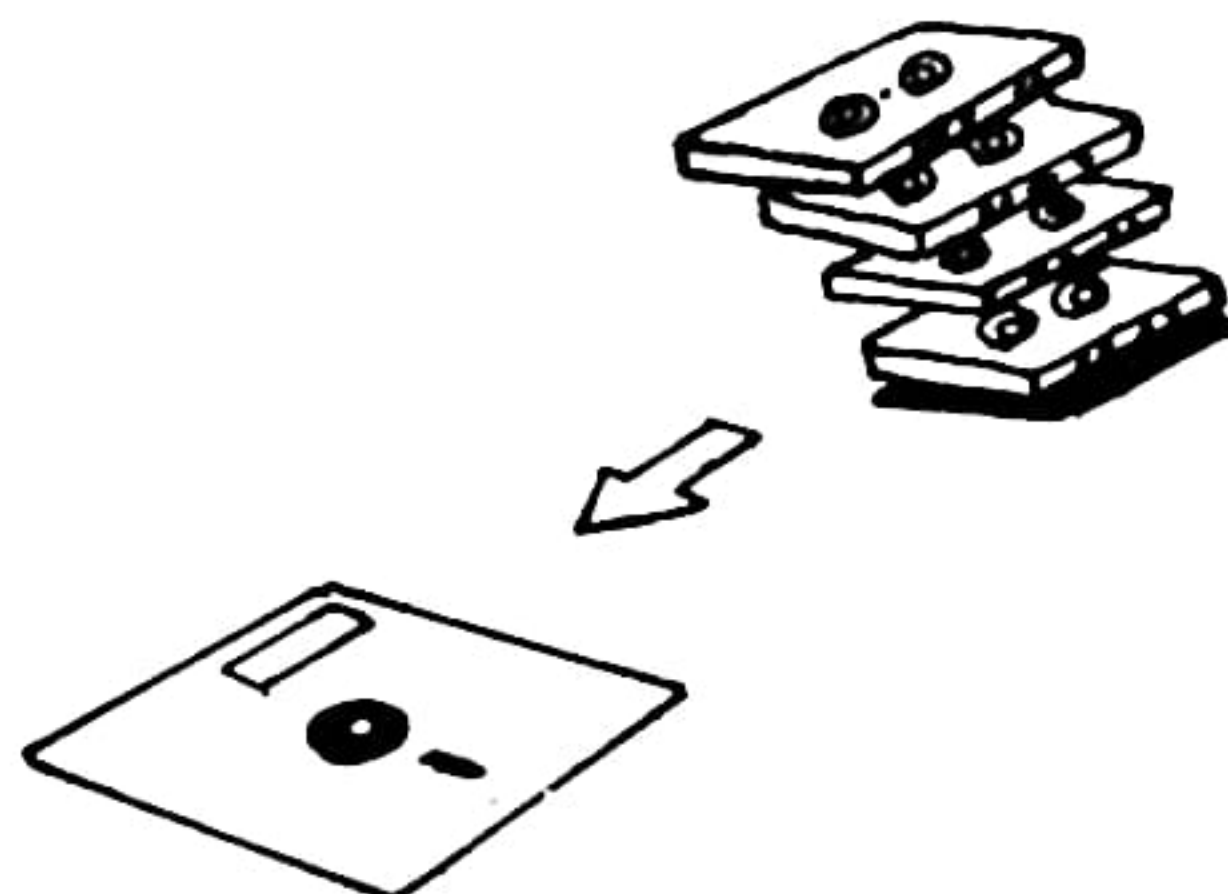
C D E A2(N、2)

1レコード長

片面45分の90分テープで約15レコード録音できた。

ころんで送るということじゃないのよ。データを送る時は転送という言い方をするんです。

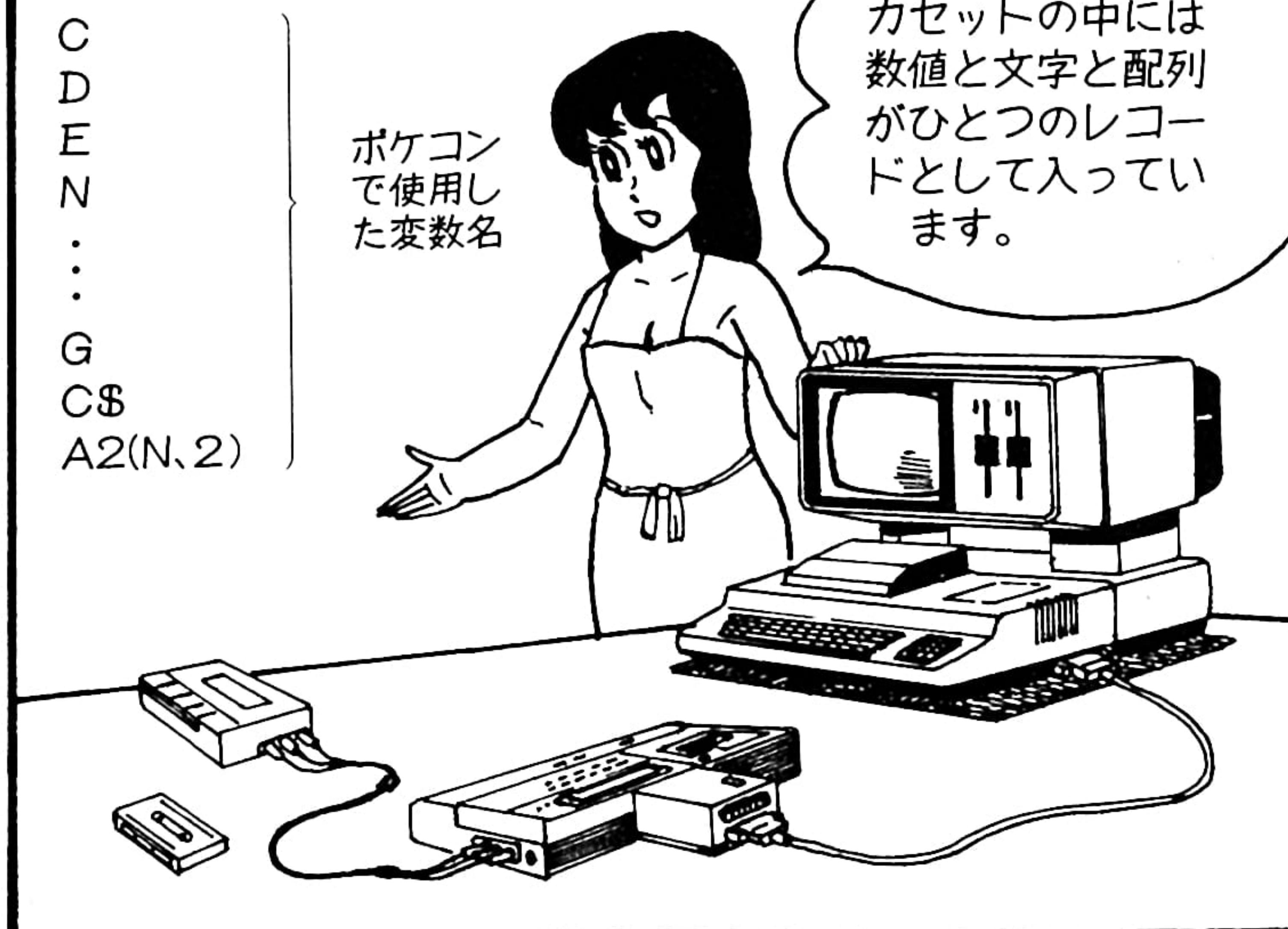
ここではカセットの中のデータをポケコンに呼び出して変数の中に入れそれをRS-232Cでパソコンに送る手続きをやってみます。

データ

表示

そこで最終的にはカセットに作ったファイルをフロッピー(FD)に移しかえる手続きが必要になると思います。

ただ、カセットのファイルではデータの集計や編集に時間がかかったり何回もカセットを交換しなければならなかったりして大変です。

C
D
E
N
・
・
・
G
C$
A2(N、2)

ポケコンで使用した変数名

カセットの中には数値と文字と配列がひとつのレコードとして入っています。

パソコン側では送られたデータをCRTに表示します。ランダムファイルの作り方は、A章でお話していますのでここでは省略します。

ヘッダー'D1'から数えて1番目のデータをCに入れ2番目のデータをDに入れ…ということなのです。

このカセットに入れているデータのフォーマットです。ヘッダー'D3'に続くデータはN×(1+2)個からなる配列要素です。

ヘッダー　C D E N B R S U　ヘッダー　C$ F G　ヘッダー　A2(N,2)

D1　D2　D3　-----

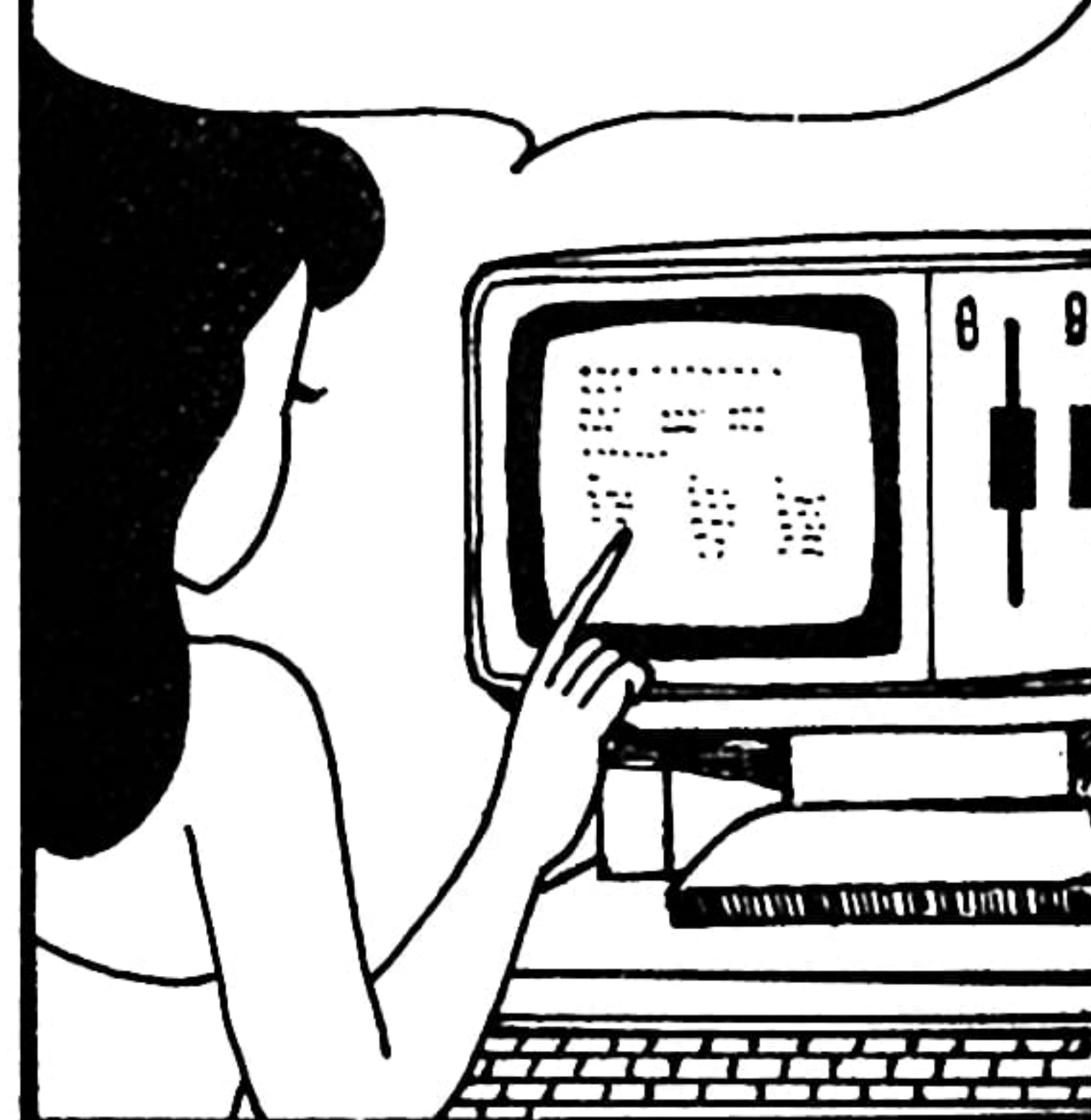
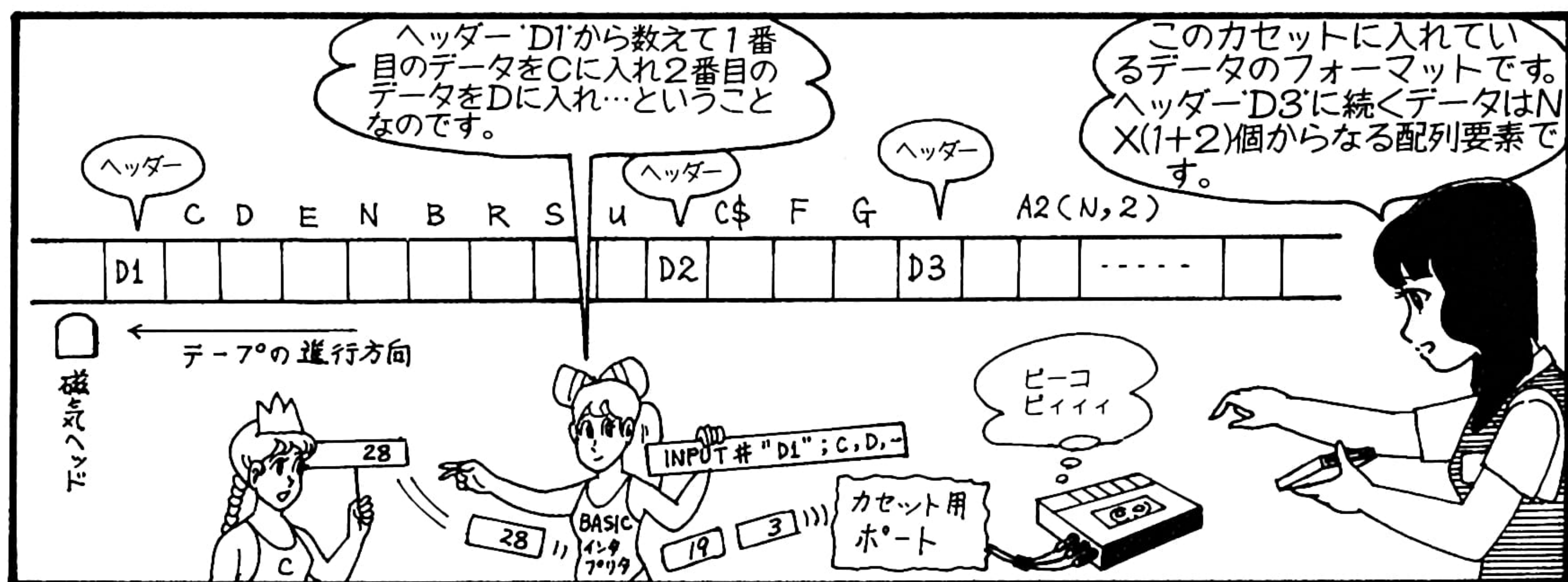
テープの進行方向

磁気ヘッド

ピーコ　ピィィィ

28　INPUT#"D1";C,D,–

BASIC インタプリタ

カセット用ポート

こちらはIF800-30側のプログラムリストです。必要なデータの先頭がどれなのか決めるためにヘッダー'ABC'を使いました。ヒントはシャープのインターフェースCE-158のマニュアルの記事から得ました。

```
10 'S59/01/17 FILE NANE RS232C3
20 OPEN "COM1:3N71" AS 1
30 PRINT #1,"ABC"
40 CLS:PRINT "PC-1500 TO IF800-30 BY RS-232C"
50 INPUT #1,Y$:PRINT Y$
60 IF RIGHT$(Y$,3)="END" THEN 260
70 IF RIGHT$(Y$,3)<>"ABC" THEN 50
80 PRINT #1,"ABC"
90 INPUT #1,C,D,E,N,B,R,S,U,F,G
100 PRINT USING "C=####,    D=####,    E=####";C,D,E
110 PRINT USING "N=####,    F=####,    G=####";N,F,G
120 INPUT #1,C$
130 PRINT "C$=";C$
140 IF Z=1 THEN 160
150 DIM A2(N,2):Z=1
160 FOR J=0 TO 2
170 FOR I=1 TO N
180 INPUT #1,A2(I,J)
190 NEXT I
200 NEXT J
210 PRINT "I","J=0","J=1","J=2"
220 FOR I=1 TO N
230 PRINT I,A2(I,0),A2(I,1),A2(I,2)
240 NEXT I
250 X=1:FOR I=1 TO 15000:X=X*X:NEXT I:GOTO 40
260 PRINT "PC-1500 TAPE END":END
```

これはポケコン側のプログラム・リストです。カセットからデータを読み込み、それをIF-800-30に転送するものです。

```
 10:REM PC-1500 TO
     IF800-30  BY
    RS-232C
 20:REM S59/01/09
 95:WAIT 50:Z=0
100:INPUT #"D1";C,
    D,E,N,B,R,S,U
120:INPUT #"D2";C$
    ,F,G
140:IF Z=1THEN 200
160:DIM A2(N,2):Z=
    1
200:INPUT #"D3";A2
    (*)(214ページ参照)
300:SETCOM 300,7,N
    ,1
305:OUTSTAT 0
310:"A":SETDEV PO:
    IF N<>1THEN 32
    0
315:LPRINT "END":
    WAIT :PRINT "T
    APE END":END
320:LPRINT "ABC"
325:WAIT 10
330:SETDEV KI:
    INPUT Y$:PRINT
    Y$:IF RIGHT$ (
    Y$,3)<>"ABC"
    THEN 330
350:SETDEV PO
400:LPRINT C,D,E,N
    ,B,R,S,U,F,G
420:LPRINT C$
430:FOR J=0TO 2
440:FOR I=1TO N
450:P=A2(I,J)
460:LPRINT P
470:NEXT I
480:NEXT J
490:CLS :GOTO 100
```

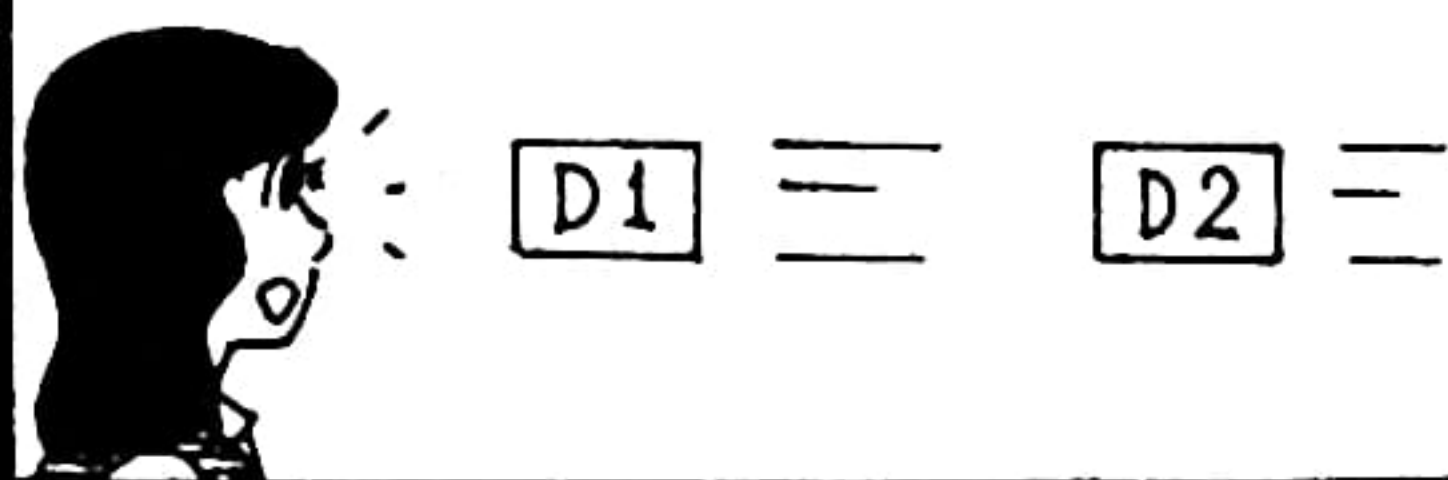
'D3'に続くデータの配列です。要素の大きさに変数Nを使っているので'D3'を読み込む前にDIM(N,2)を宣言する必要があります。文140は2回以降再宣言するのをさけるためのものです。

```
140: IF Z=1 THEN 200
160: DIM A2(N,2):Z=1
200: INPUT # "D3"; A2(*)
```

DIM A2(N,2)　D3

まずポケコン側からこのリストの意味を説明しましょう。テープからの読み込み命令はINPUT#、このリストではヘッダー'D1'、'D2'、'D3'をつけてデータの先頭を検出するようにしています。

```
100: INPUT # "D1"; C,D,E,
       N,B,R,S,U
120: INPUT # "D2"; C$,F,G
```

D1　D2

SETDEV PO(セットデブポートアウト)は、LPRINT(エルプリント)文で出力するためのものです。

```
SETDEV PO
   :
LPRINT
   :
SETDEV
   :
LPRINT
   :
```

RS-232Cへの出力文となる

RS-232Cへの入出力の解除

プリンタへの出力文となる

'A'はラベルです。DEF Aのキー操作でプログラムがここから走ります。この、ラベルを使ってのプログラムスタートでは数値がクリアされないのでテストの時便利ですよ。

```
310:"A":SETDEV PO
```

データ1レコード分をポケコンの変数に入れ終わったら、次は転送の用意です。まず、コミュニケーション条件のセットです。

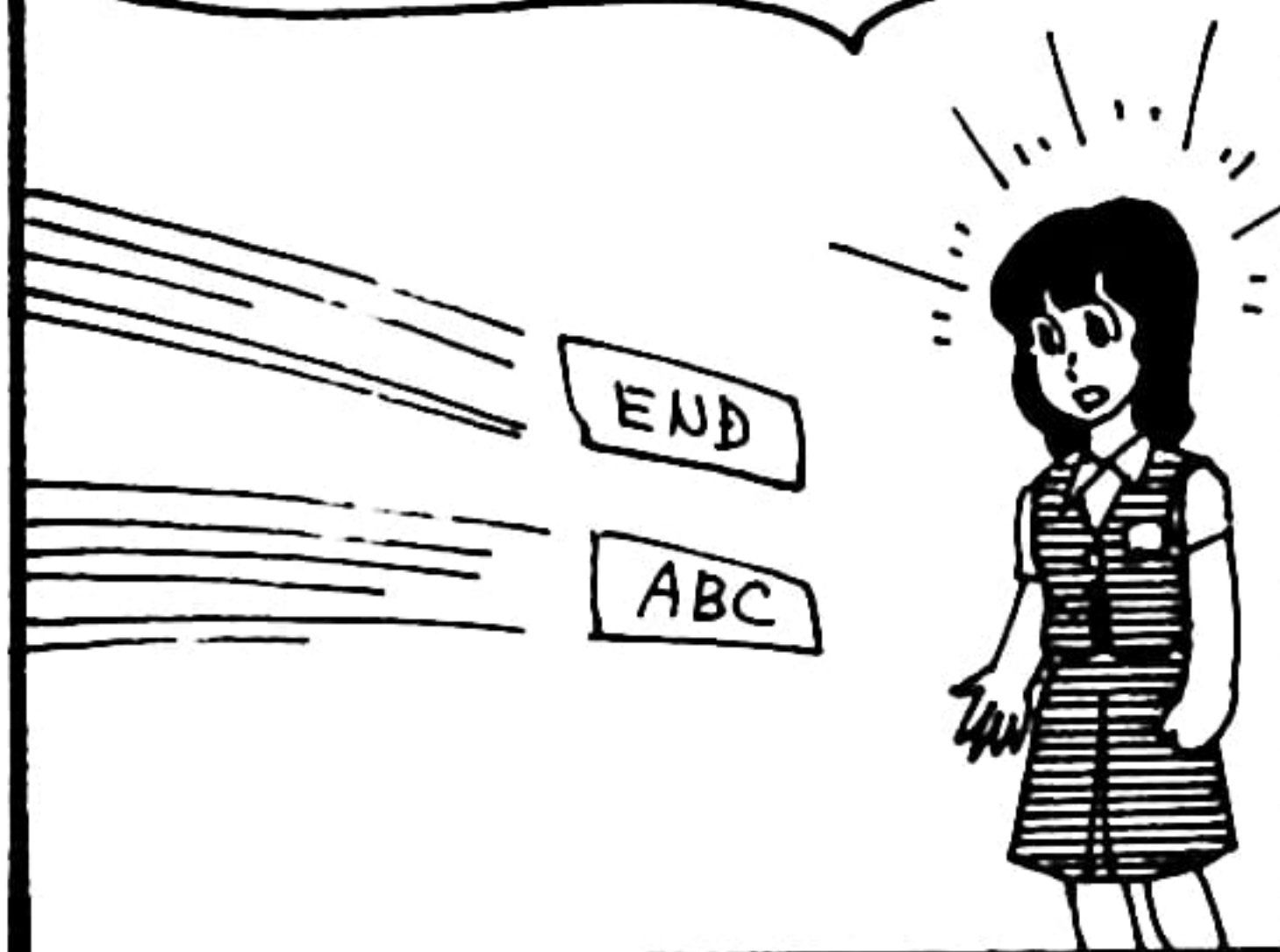
```
300:SETCOM 300,7,N,1
305:OUTSTAT 0
```

プログラム終了ルーチンです。RS-232Cポートに'END'を出し、TAPE END'と表示します。ENTERを押すと、>(プロンプト表示)になってプログラム終了です。

```
315:LPRINT "END":WAIT:
    PRINT "TAPE END":
    END
```

'END'は、IF側にプログラム終了を知らせるためのキーワード。'ABC'はデータ転送に先がけて送るヘッダーです。

END

ABC

カセットの最終レコードにはNに1を入れてあります。N<>1はNが1でなかったら、という意味です。この<>(でなかったら…)は便利です。

```
IF N<>1 THEN 320
```

もしもNが1でなかったら320へ、1なら次の文へ…

ポケコンからの'END'または'ABC'を受けとるIF800-30側のプログラムです。

```
50 INPUT #1,Y$ :PRINT Y$
60 IF RIGHT$(Y$,3)="END"THEN 260
70 IF RIGHT$(Y$,3)<> "ABC" THEN 50
80 PRINT #1,"ABC"
```

その対策としてはやっぱりヘッダーを送るしかないなと思ったの。

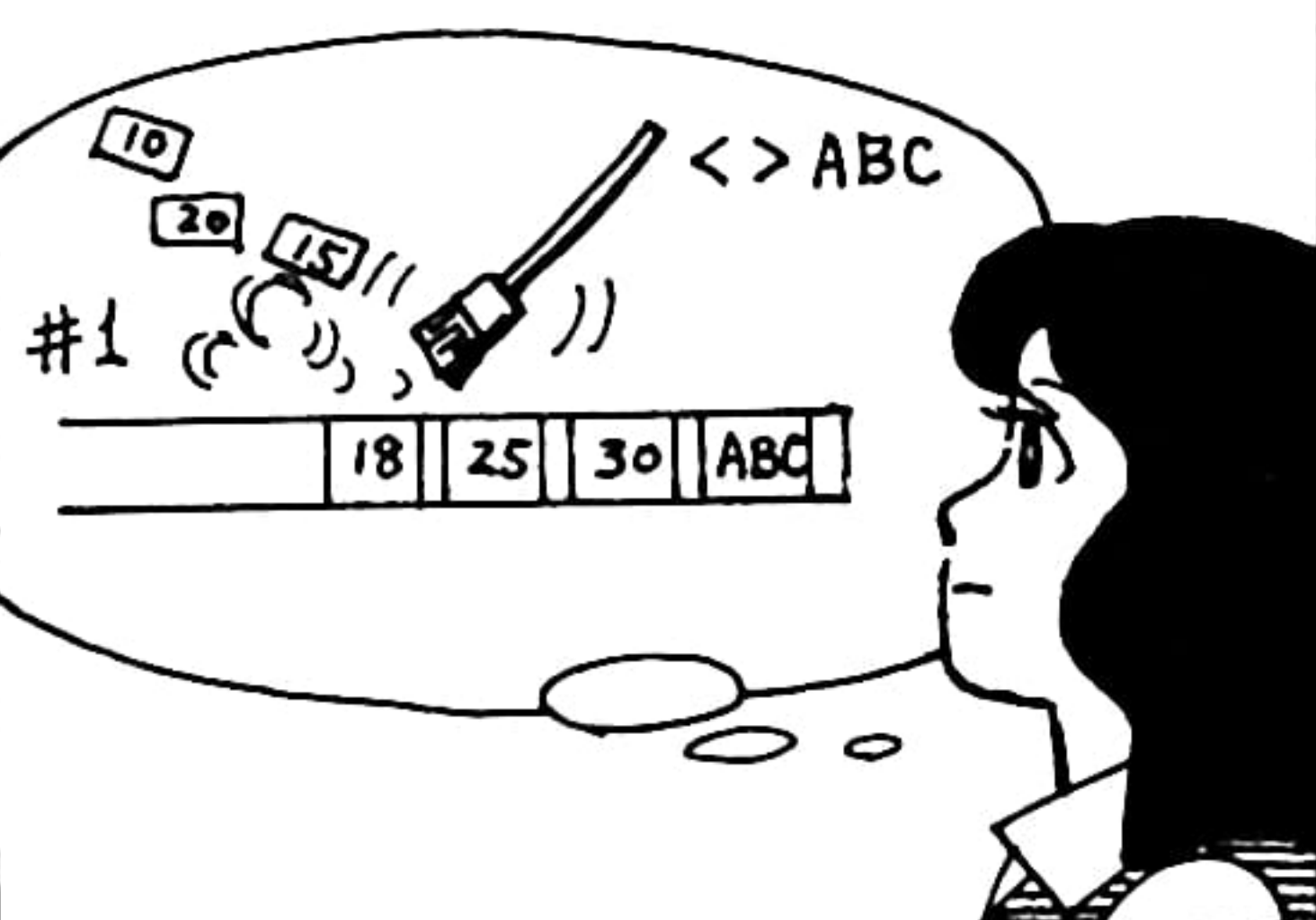
このヘッダーは別にABCでなくてもよいのですが、マニュアルを参考にしましたのでそのまま使いました。

```
320:LPRINT "ABC"
```

途中でプログラムをキャンセルした時などバッファの中にデータが残ることがあるのよね。

バッファの中身が何かはわかりませんからRIGHT $(Y$,3)<>'ABC'は、バッファの中身を掃除することになります。

10 20 15 <>ABC

#1 18 25 30 ABC

通信用バッファはFIRST IN、FIRST OUTです。先に入ったデータが先に取り出されます。

これが次のINPUT#1,Y$でY$に取り出されるデータです。

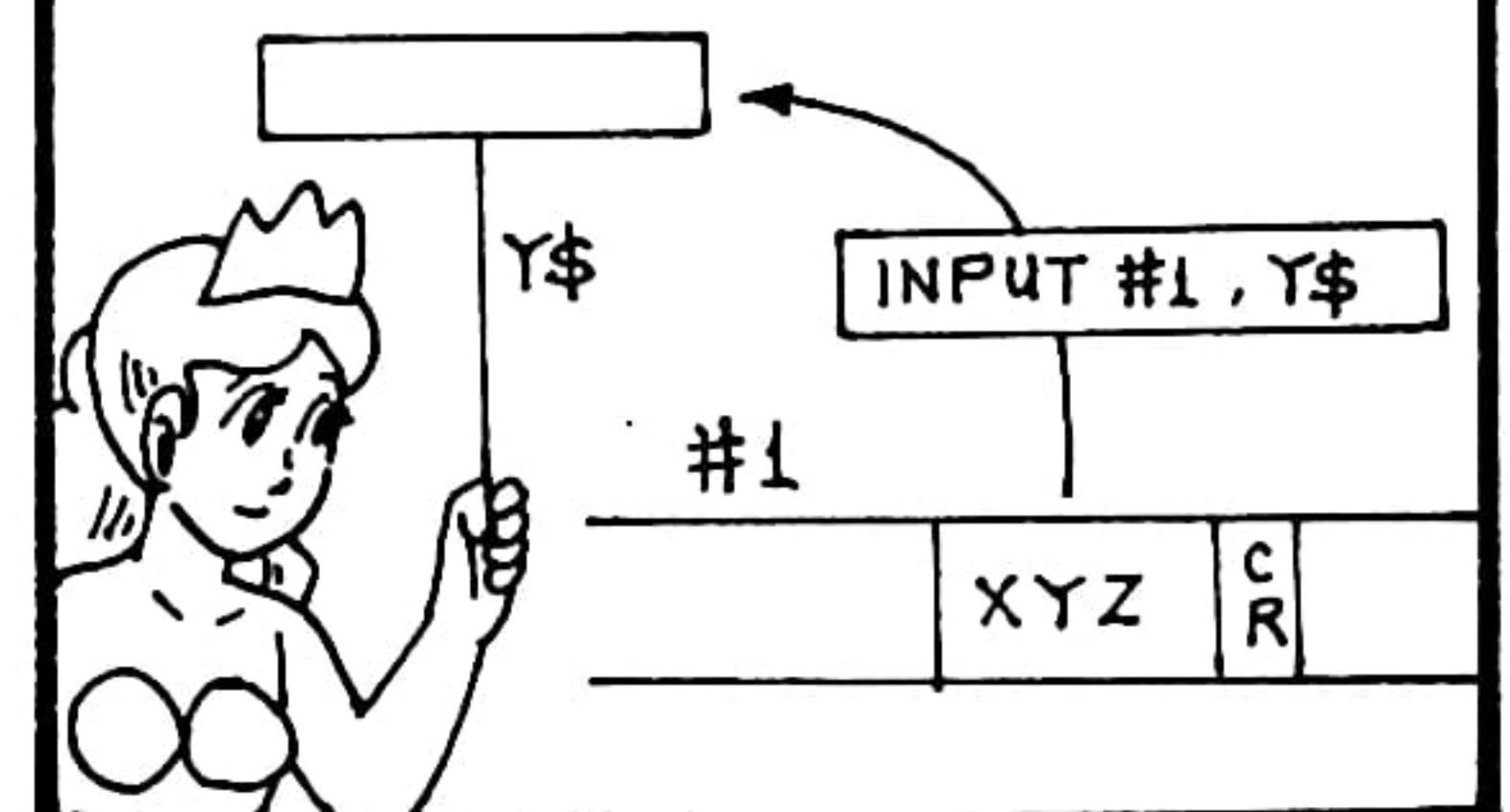
#1 20 CR 18 CR 15 CR

次のデータが入ると以前のデータは前にシフトされます。

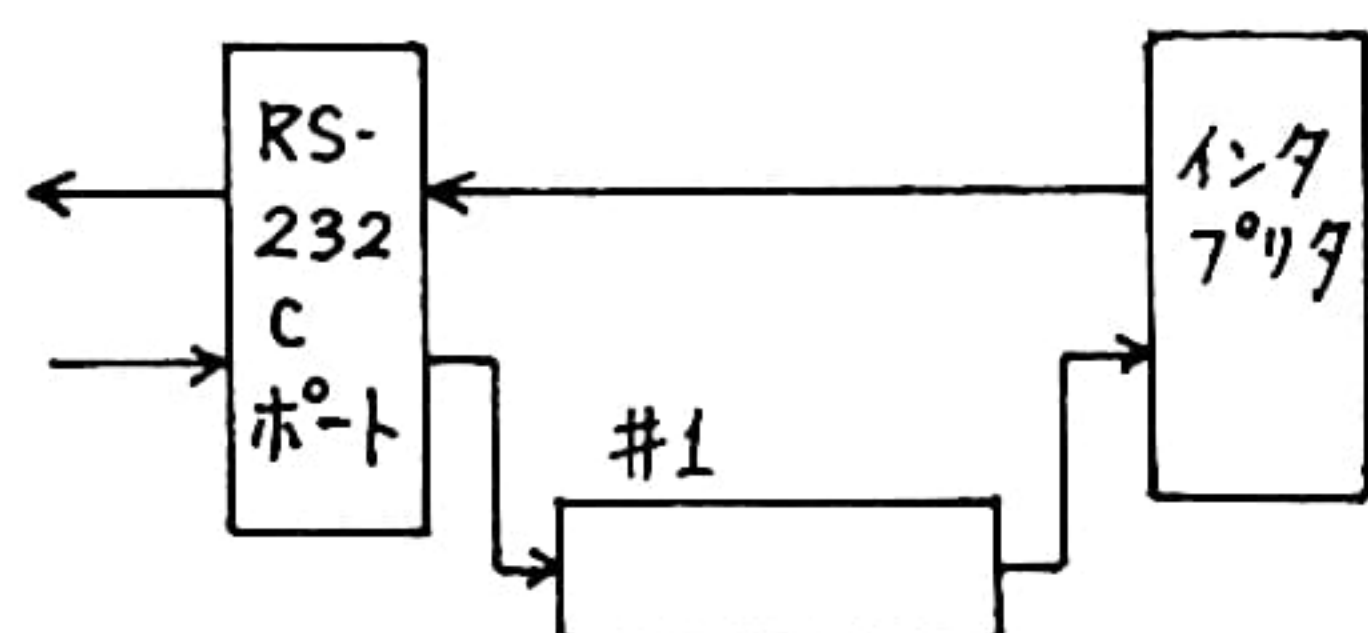
まず、INPUT#1,Y$ではバッファから先頭のデータ(最初のCRまで)を取り出し変数Y$に入れます。取り出されたデータはバッファから消えます。

Y$

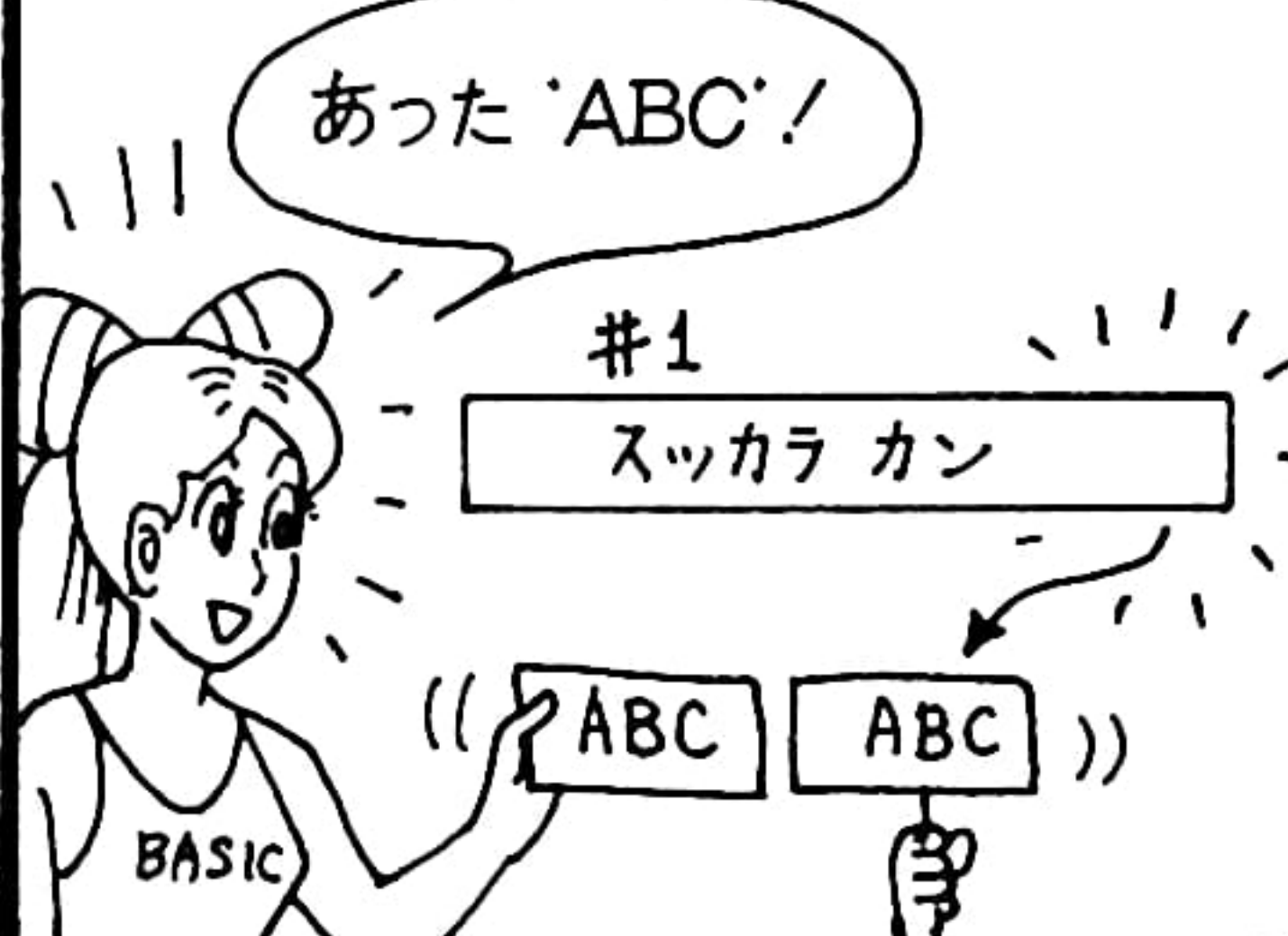
INPUT #1,Y$

#1 XYZ CR

そこでIF800-30からも、キーワードとして'ABC'を返してやります。

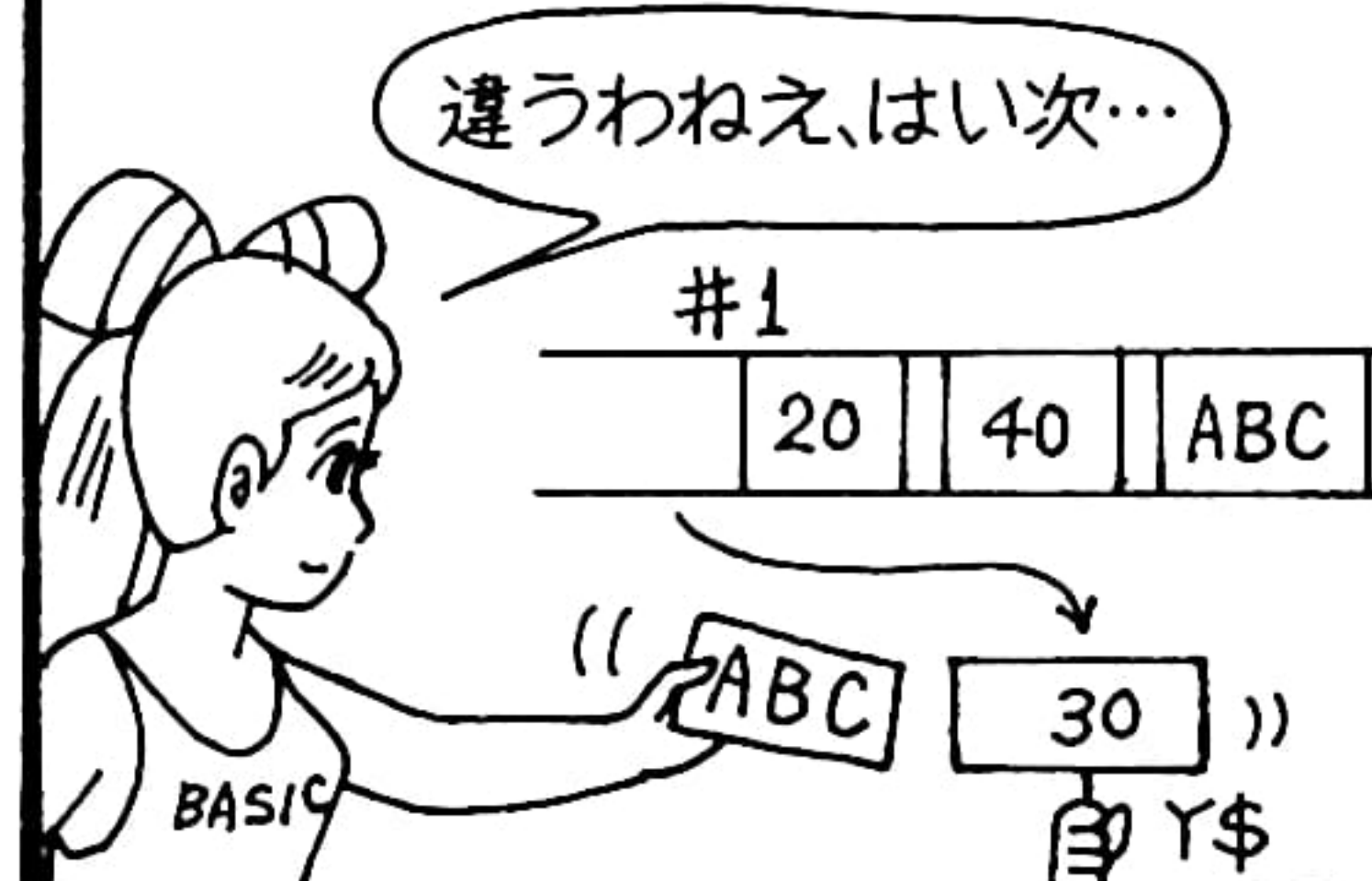
```
80: PRINT #1, "ABC"
```

RS-232C ポート
インタプリタ
#1

ポケコン側からはヘッダー'ABC'が送られただけですから、バッファから'ABC'が取り出された時点でバッファの中はカラッポになってしまいます。

あった'ABC'!

#1
スッカラ カン
ABC ABC
BASIC

それはY$に'ABC'が入るまでバッファからデータを取り出しては捨てているからです。つまりヘッダー'ABC'以前のバッファの中身は不要というわけです。

違うわねえ、はい次…

#1
20 40 ABC
ABC 30
BASIC
Y$

これで送受信側とも準備完了というわけです。RS-232Cには先に説明した制御信号がありますが操作法がわからないし、あてになりません。ソフトで工夫した方が確かです。

ポケコン側ではヘッダー'ABC'を送ったあと、SETDEV KIとして、INPUT Y$に'ABC'が来るのを待っています。この場合もポケコン側のバッファが掃除されることになっています。

```
330: SETDEV KI : INPUT Y$ : PRINT Y$:
     IF RIGHT$ (Y$, 3) <> "ABC" THEN 330
```

その次が配列の中身です。一足先に送ったNを使って配列を宣言する必要があります。

```
430: FOR J=0 TO 2
440: FOR I=1 TO N
450: P=A2(I,J)
460: LPRINT P
470: NEXT I
480: NEXT J
```

どうして数値変数と文字変数を分けたかというと、INPUT #1のあとに両方混合したものをつけると表示がおかしくなったからです。

```
ポケコン
  LPRINT C$, F, G
     ↓
IF800-30
  INPUT #1, C$, F, G
```

次にポケコン側は、まず数値変数の中のデータをまとめて出力します。次に文字変数の中身を出力します。

```
400: LPRINT C, D, E, N, B,
     R, S, U, F, G
420: LPRINT C$
```

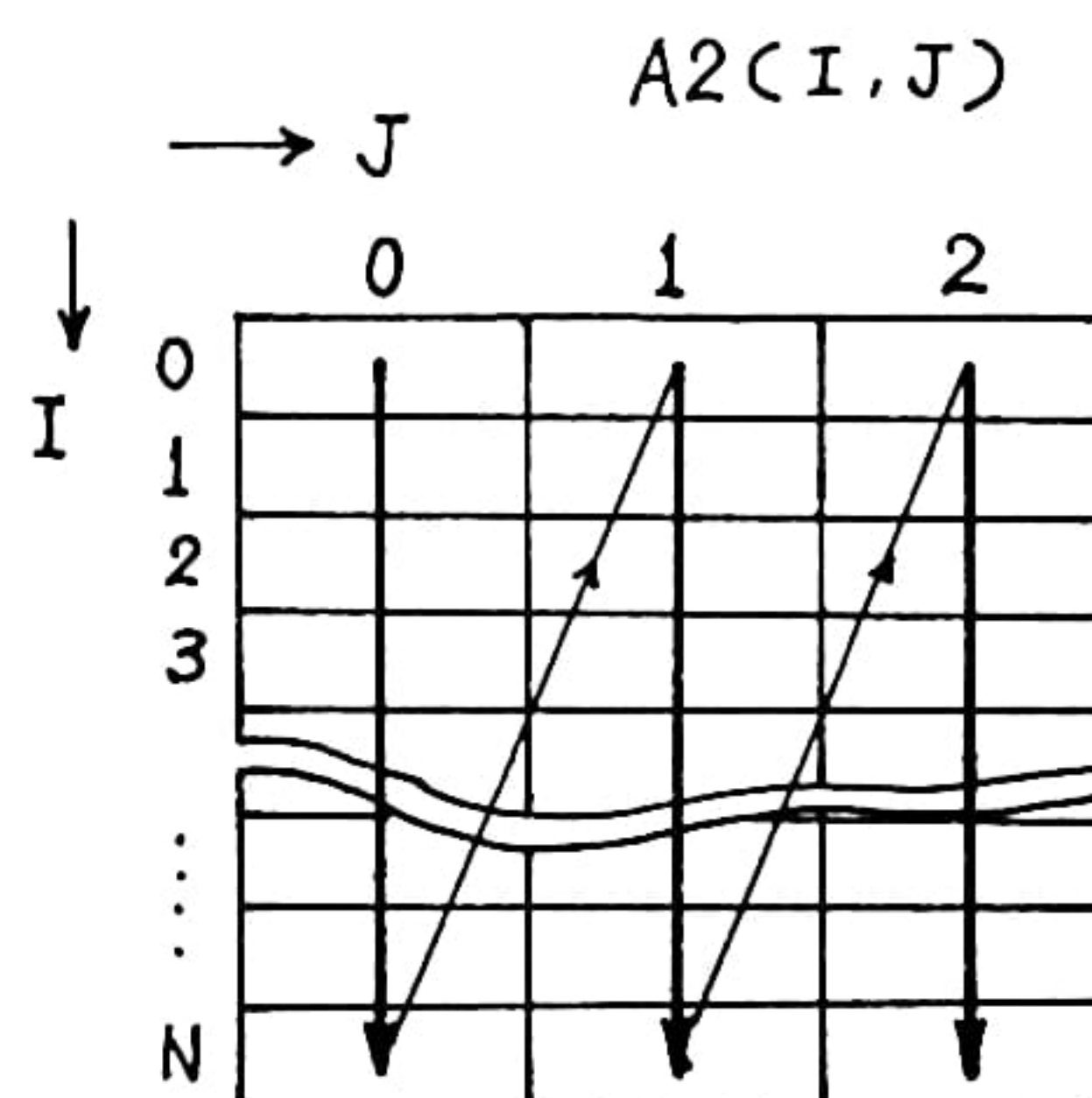
IF800-30側でもポケコン側が送り出したのと同じ順序でバッファから各変数に代入してやらなければなりません。

```
90  INPUT #1, C, D, E, N, B, R, S, U, F, G
 :
120 INPUT #1, C$
 :
150 DIM A2(N, 2) : Z=1
160 FOR J=0 TO 2
170 FOR I=1 TO N
180 INPUT #1, A2(I, J)
190 NEXT I
200 NEXT J
```

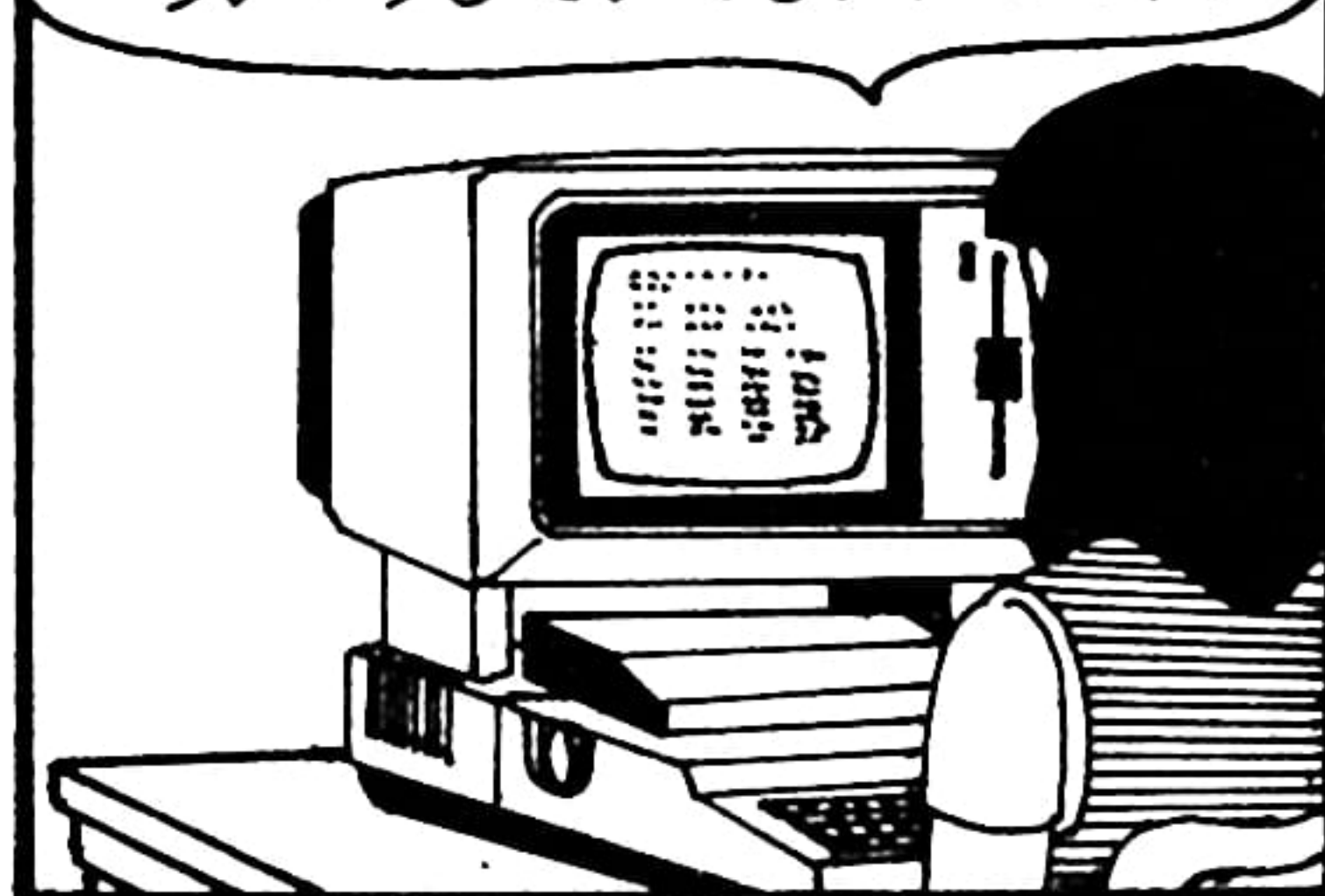
どこに何が入ってくるかさえわかっていれば別に同じ変数名を使わなくてもいいんですよ。

IF800側では配列に直接読み込みます。

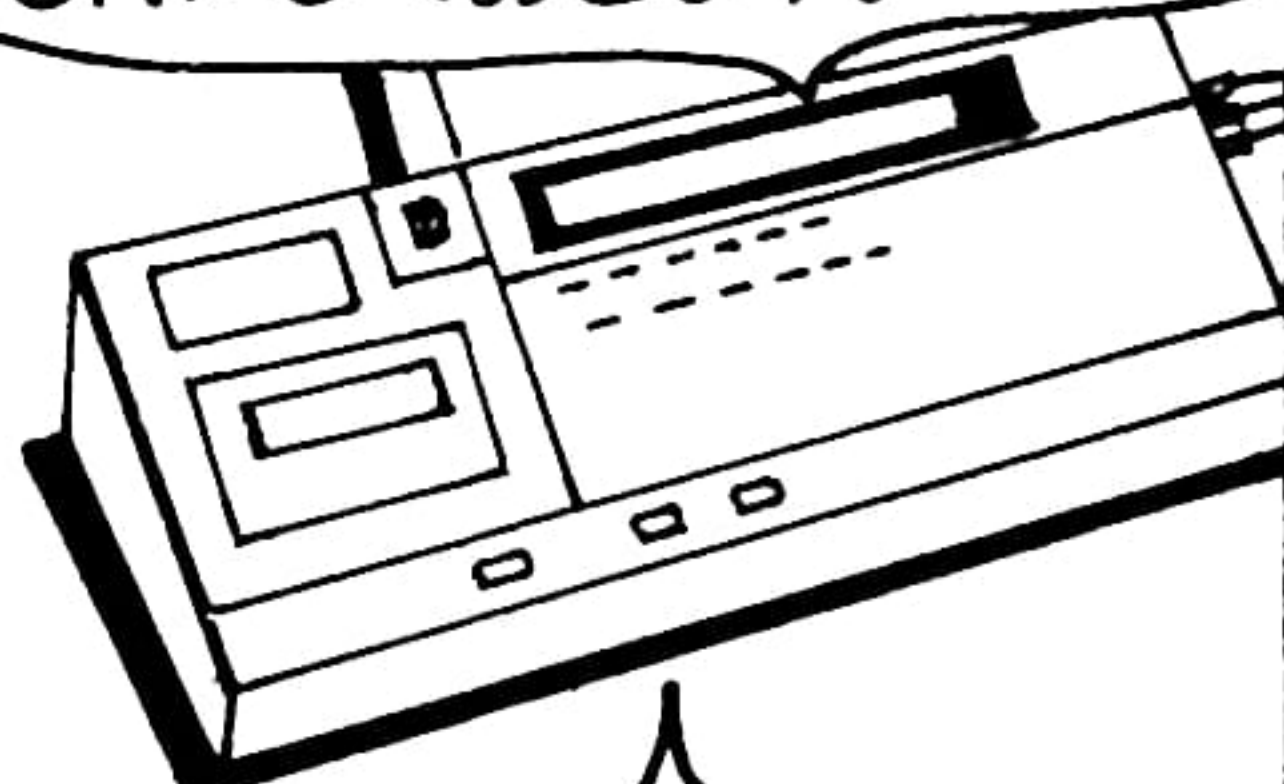
データはこの順序で出ていきます。

A2(I, J)
→ J
0 1 2
↓ I
0 1 2 3 … N

では、このシステムを動かしてみましょう。具体的な操作を教えてもらうことは慣れることにつながります。習ったことだけでは自分のものになりません。

ポケコンがカセットからデータを読んでいる間に何か別の仕事ができないか…というようなことは単発的なプログラムでは期待しない方がいいですね。

文250は表示のための時間かせぎに使っています。文40に戻るとCLSでクリアされてしまうからです。ポケコンから次のデータが送られるまで3分程度ありますので、2分ぐらいこうしていてもいいのです。

プログラムをセーブする時にも、ヘッダーをつけておくと便利です。カセットにはこのヘッダーとプログラム名とカウンターの値を記入しておきます。

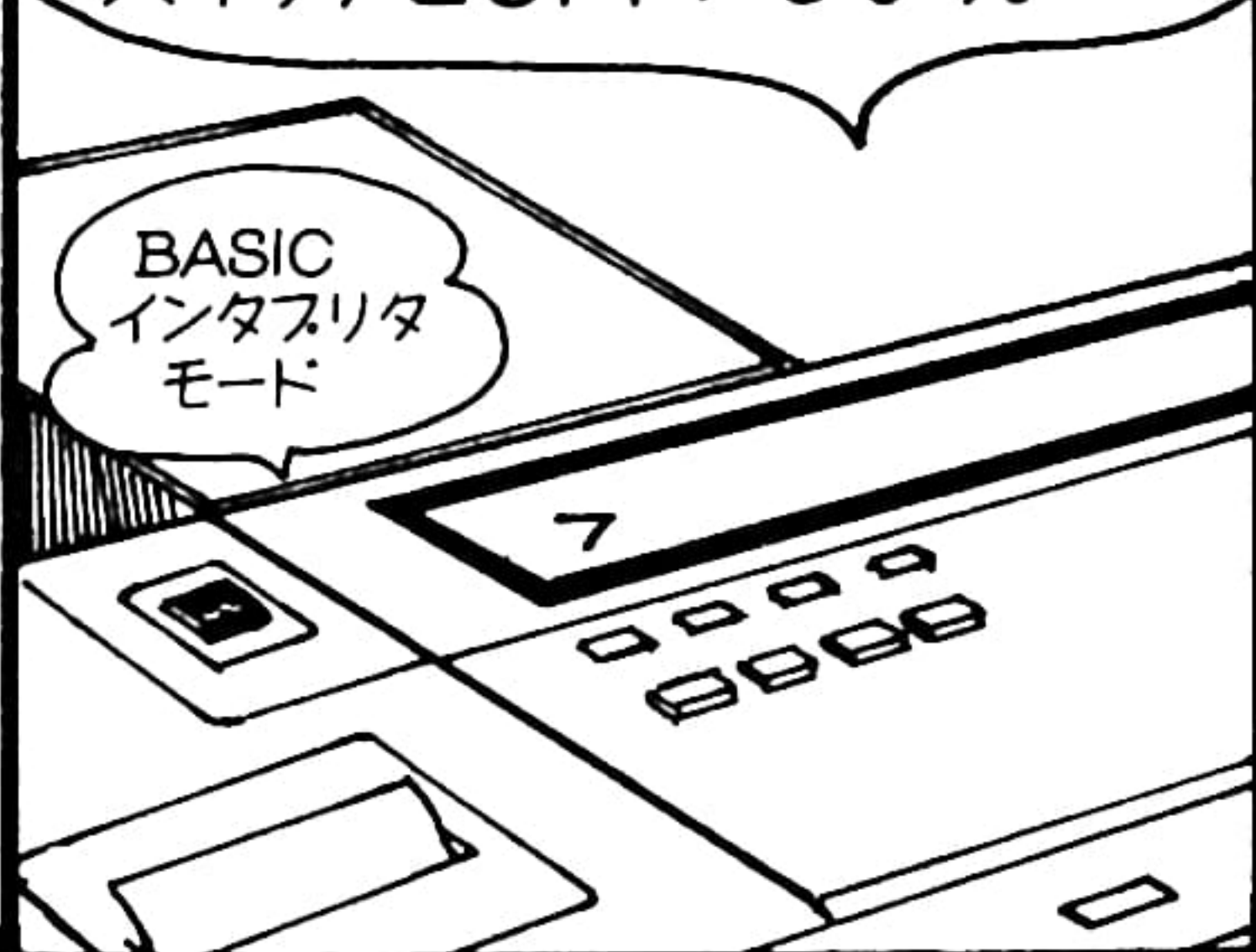
ヘッダー	プログラム名	カウンタ
A	PC1500 TO IF800-30 BY RS-232C (データカセットイン)	005 ~0013

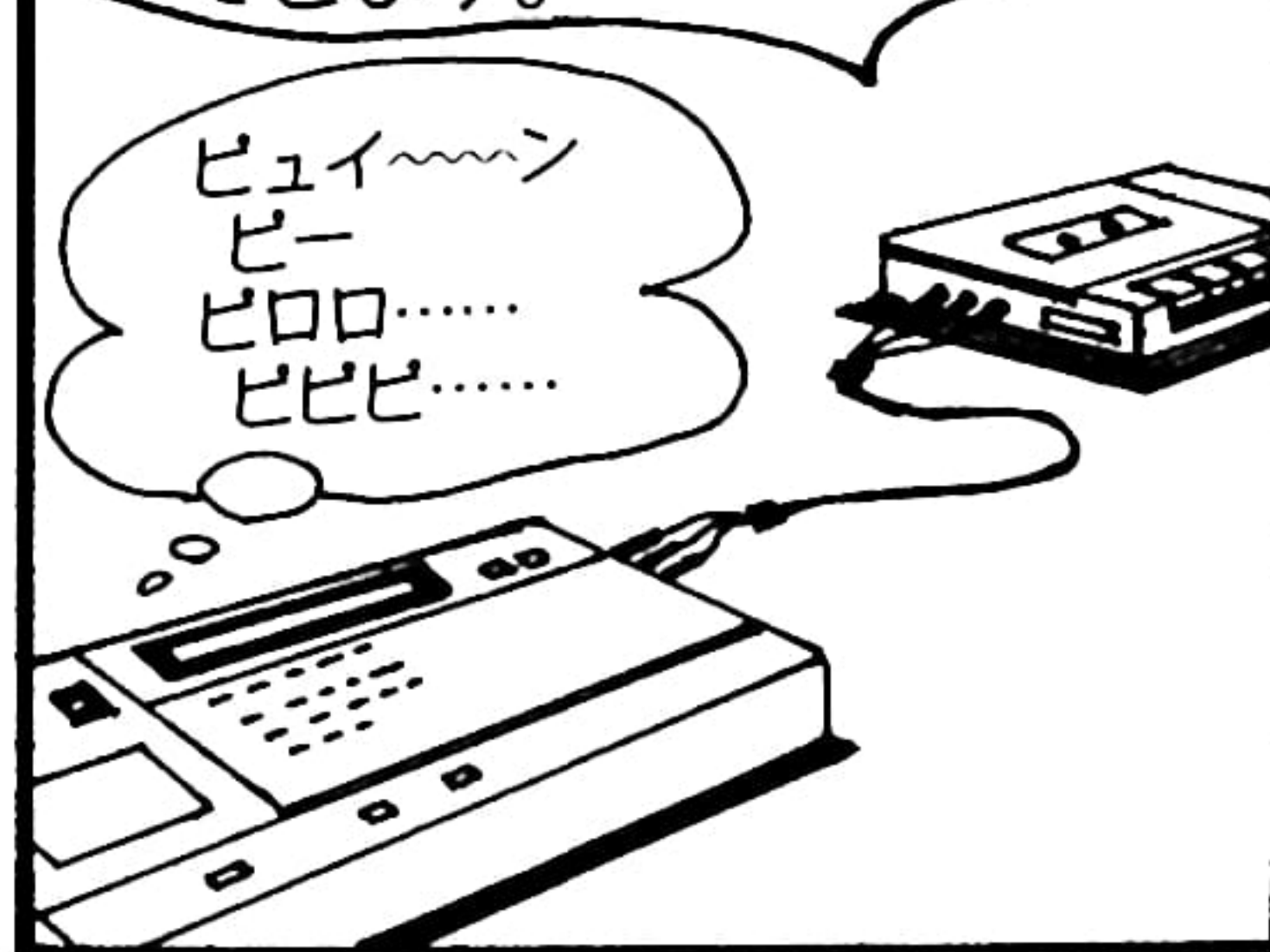
プログラムを入れたカセットをセットしPLAY(再生)スイッチをONにしておきます。

ポケコン側のリモートスイッチをONにしておかないとスイッチが回ってしまいます。

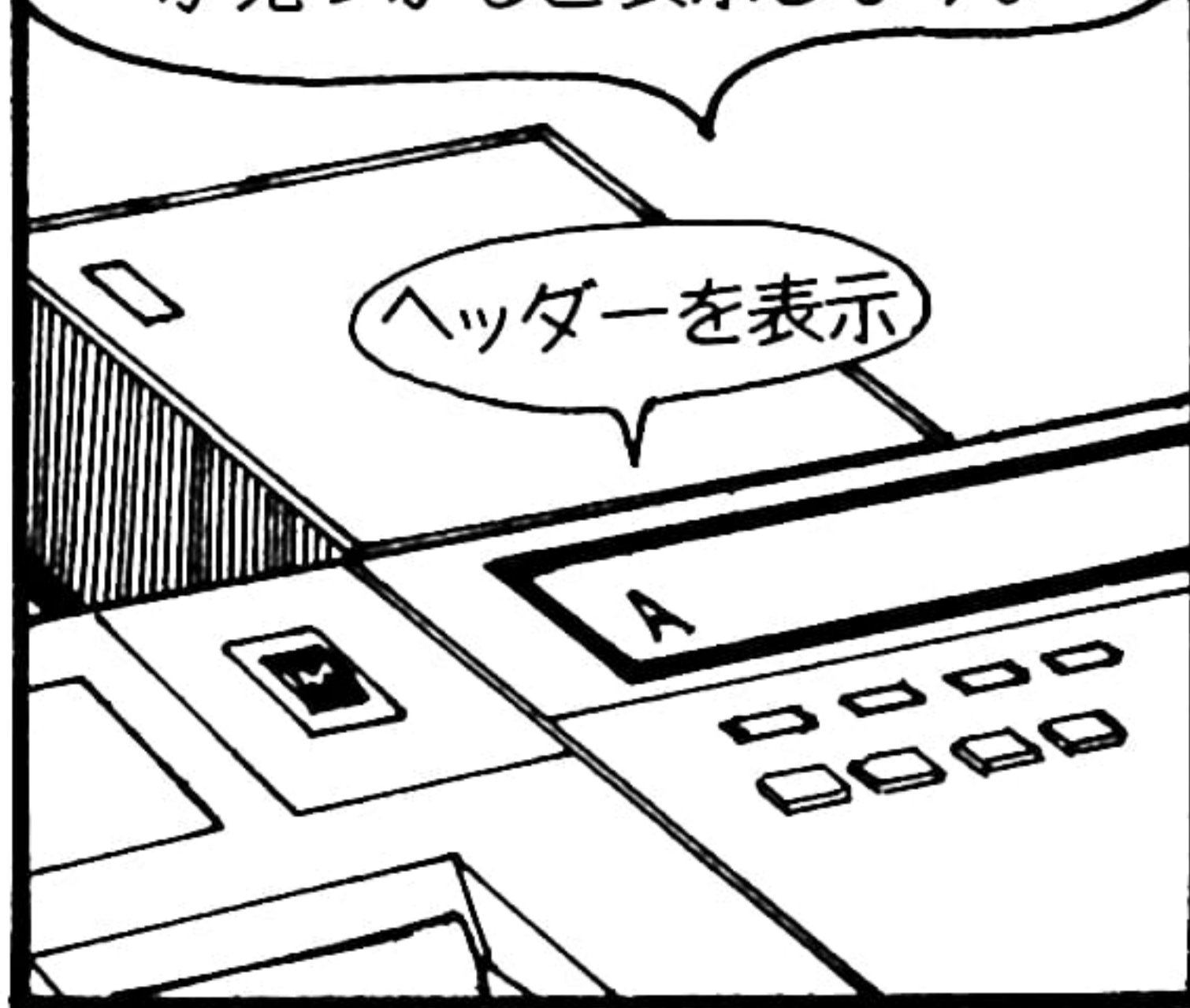
まずテープレコーダーです。ボリュームをMax(最高)にしておくのがポイントです。また、バッテリーよりもDCアダプターを使った方が確実でした。

カウンタ
リモートスイッチ端子
録音端子
再生端子
ボリューム Max
音質 high

プログラムロードが完了すると、表示は>(プロンプト)に戻りテープの回転が止まります。PLAYスイッチをOFFにします。

BASIC インタプリタ モード

この時ポケコンは再生音を出します。この音はBEEP OFF ENTER で消すことができ、BEEP ON ENTER で出すことができます。

ピュイ〰〰ン
ピー
ピロロ……
ピピピ……

CLOAD"A" ENTER でテープが回ります。ポケコンはヘッダーが見つかると表示します。

ヘッダーを表示

RUNすると、まず"ABC"が送出されますがポケコン側が起動していないのでたれ流しとなります。制御信号も効いていないことがわかりました。次のメッセージを表示してポケコンからのデータ待ちとなります。

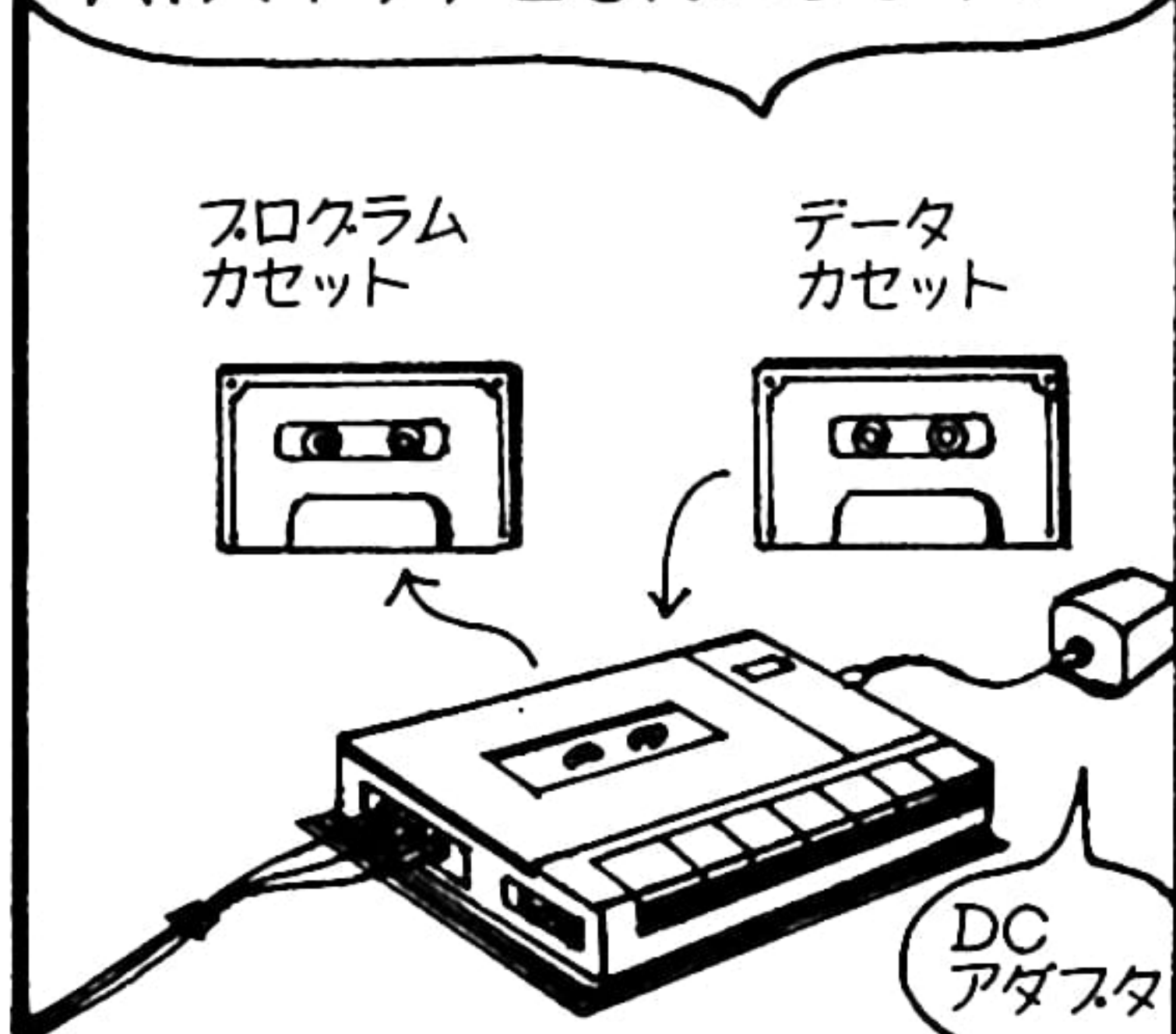
PC1500 TO IF800-30 BY RS-232C

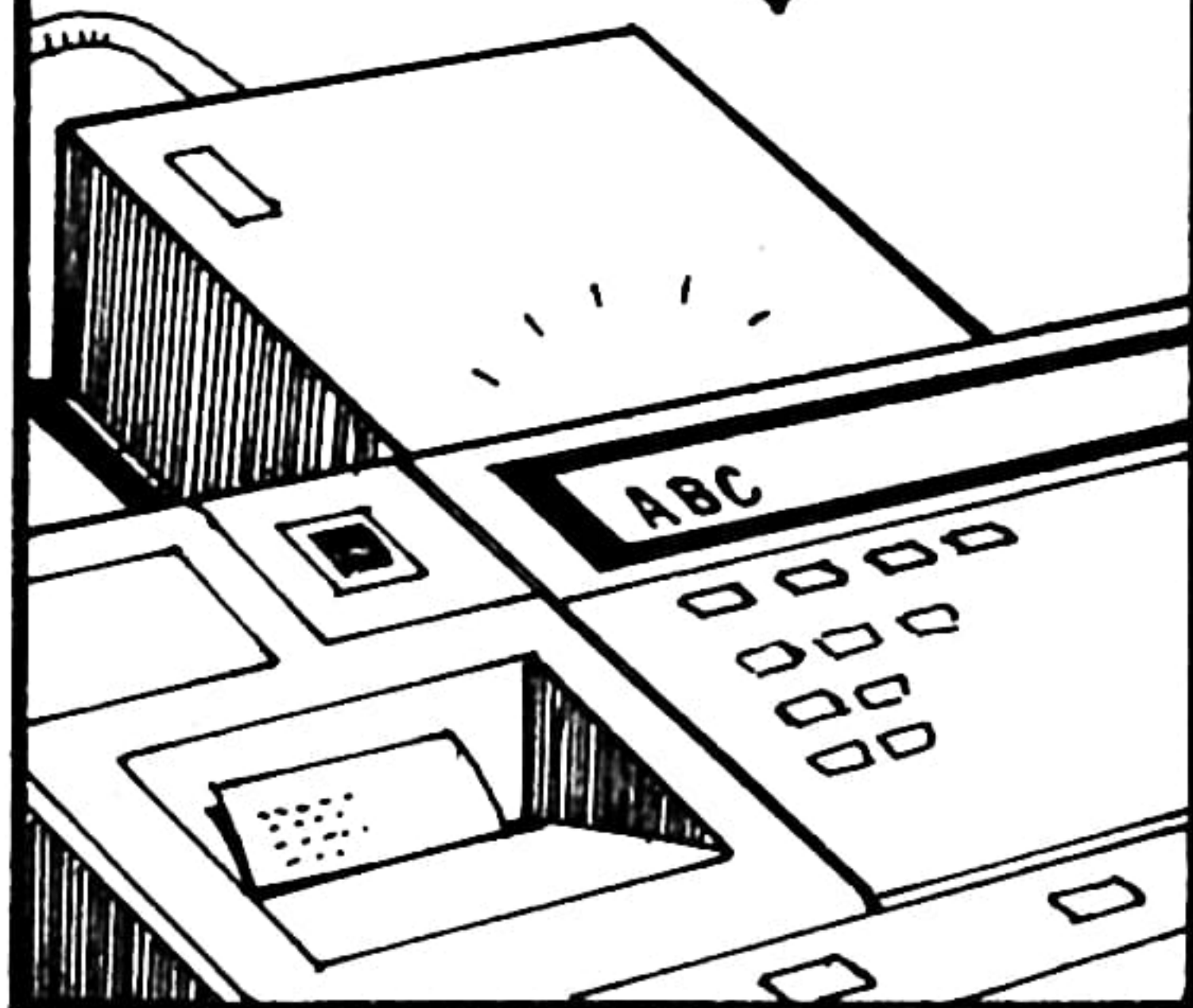
次はIF800-30側の起動です。プログラムはRS-232C1でセーブしておきますのでこれをロードします。

カムヤ
カムヤ

```
LOAD "RS-232C1"
NOT FOUND
LOAD "RS232C1"
OK
```

次にカセットをデータが入ったものに換えてセットします。PLAYスイッチをONにします。

プログラムカセット
データカセット
DC アダプタ

IF800-30は'ABC'を確認するとポケコンに'ABC'を返します。

ABC

D1、D2、D3のデータを順に読み込むとテープが止まります。そして、IF800-30に'ABC'を送り出します。

PC1500 TO IF800-30 BY RS-232C

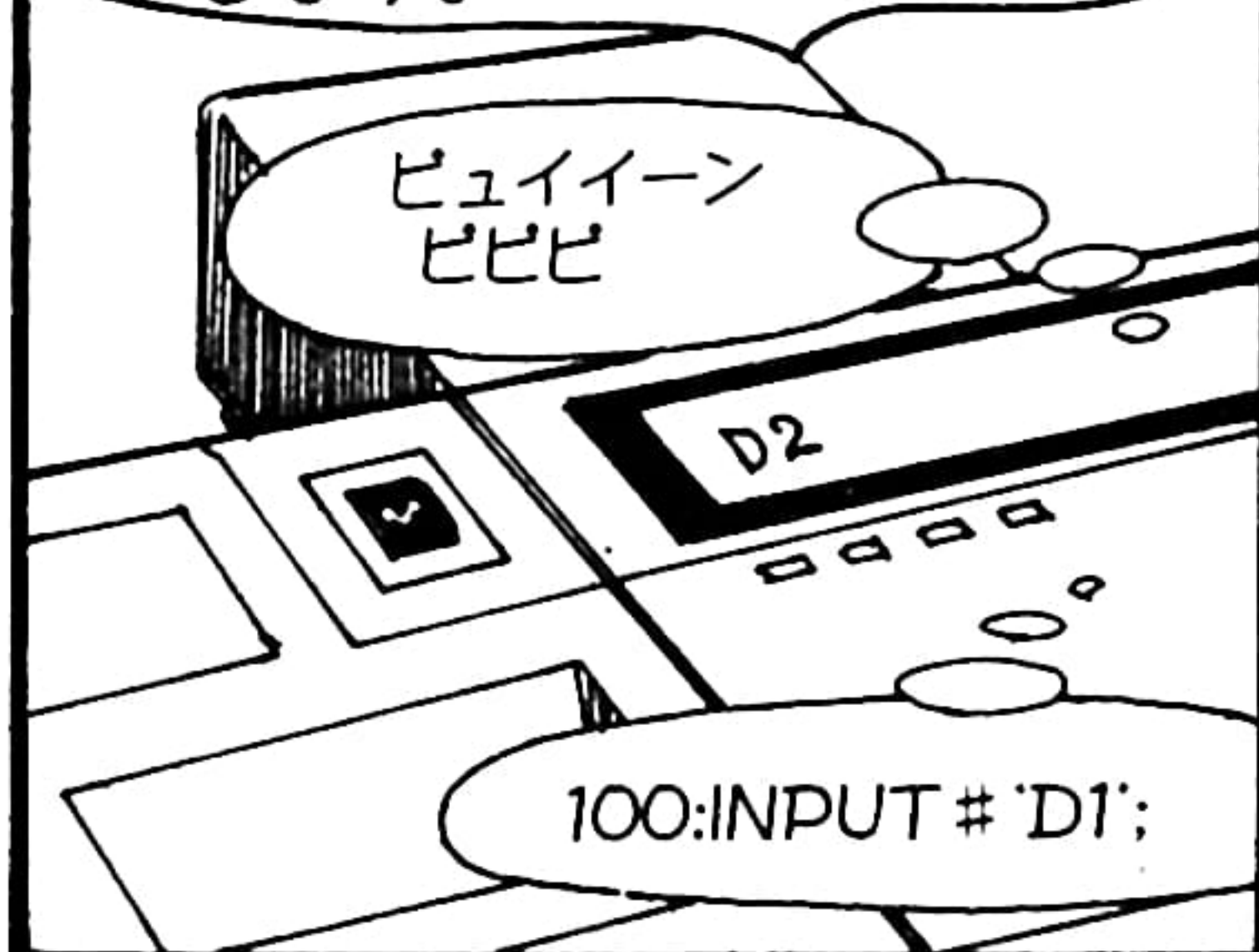
もし、バッファにデータが残っていたらここに表示されます。

ABC

きた

次にポケコン側をRUNすると、テープが回り出します。まずヘッダーD1がくるのを待っています。先にD2やD3がきても無視します。

ピュイイーン ピピピ

D2

100:INPUT #'D1';

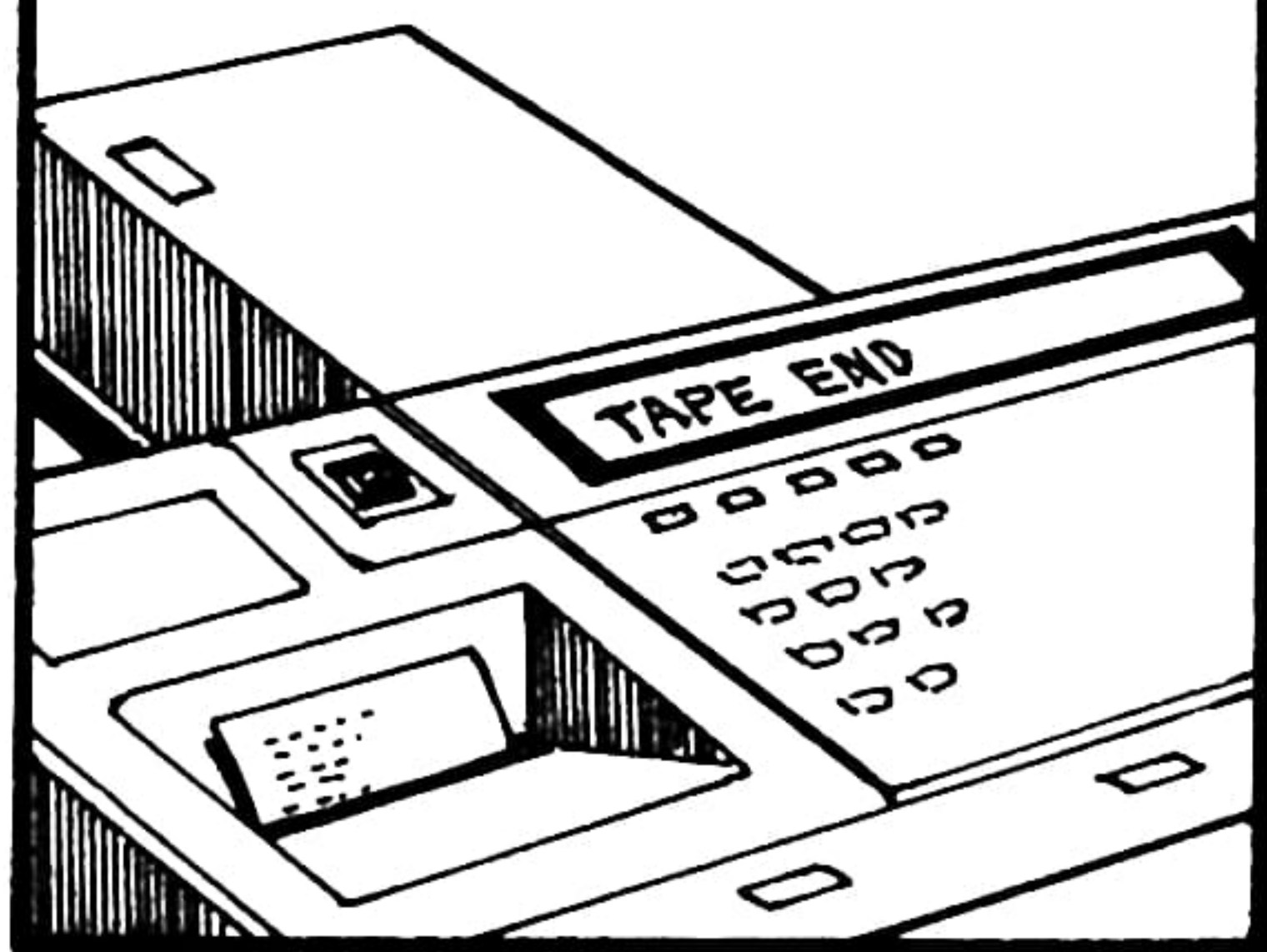
```
PC-1500 TO IF800-30 BY RS-232C
ABC
C=   10   D=   12   E=  18
N=   10   F=   56   G=   2
C$=TAPE RS232C
I          J=0        J=1        J=2
 1          10         30         15
 2          12         10         10
 3          25         15         5
 4          14         18         1
 5          23         25         9
 6          10         16         30
 7          15         10         26
 8          25         10         25
 9          17         15         14
 10         25         16         8
```

ポケコンは'ABC'を受けとると、データの転送を開始します。

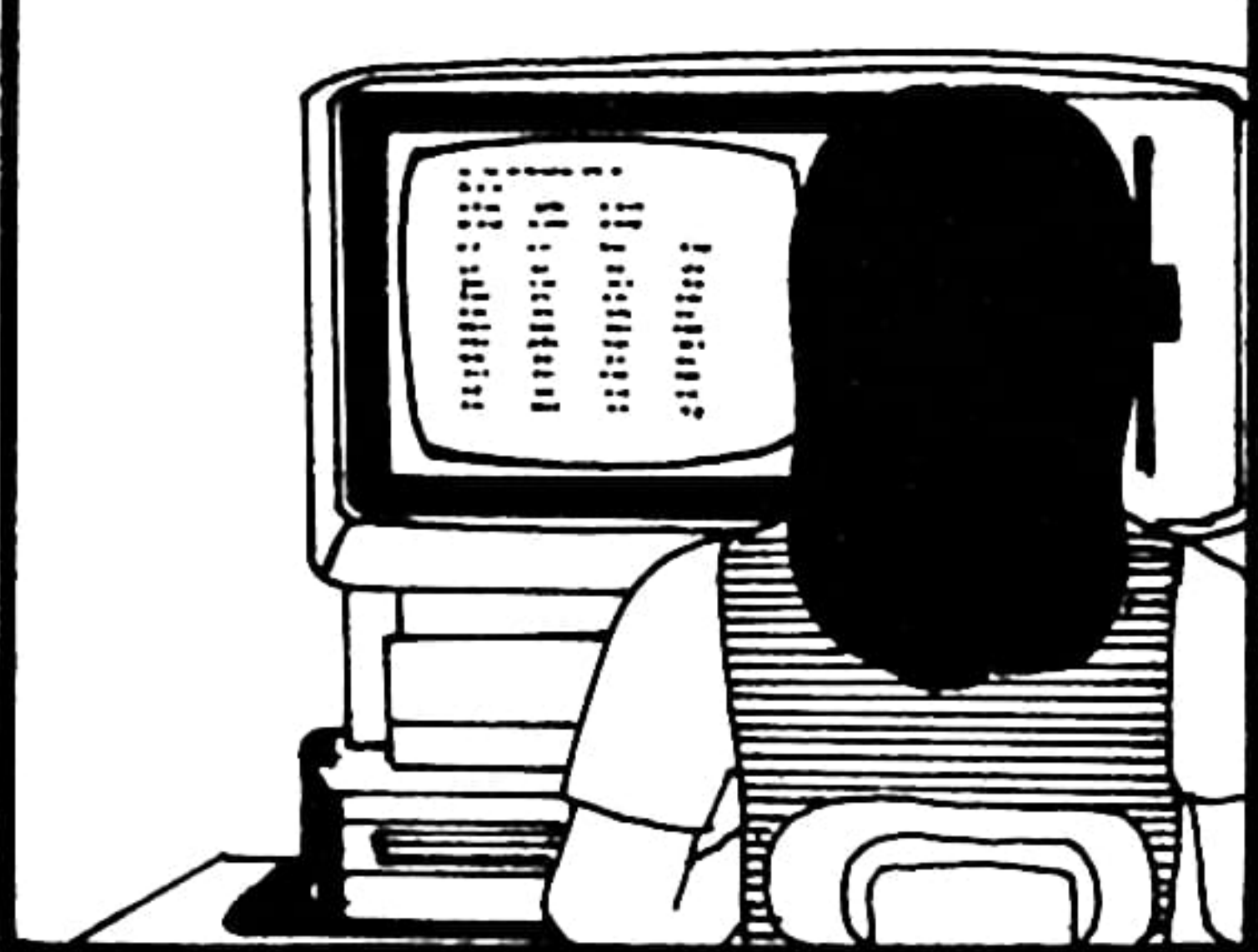
```
PC-1500 TO IF800-30 BY RS-232C
ABC
C=   10   D=   12   E=  18
N=   10   F=   56   G=   2
C$=TAPE RS232C
```

IF800-30の画面です。

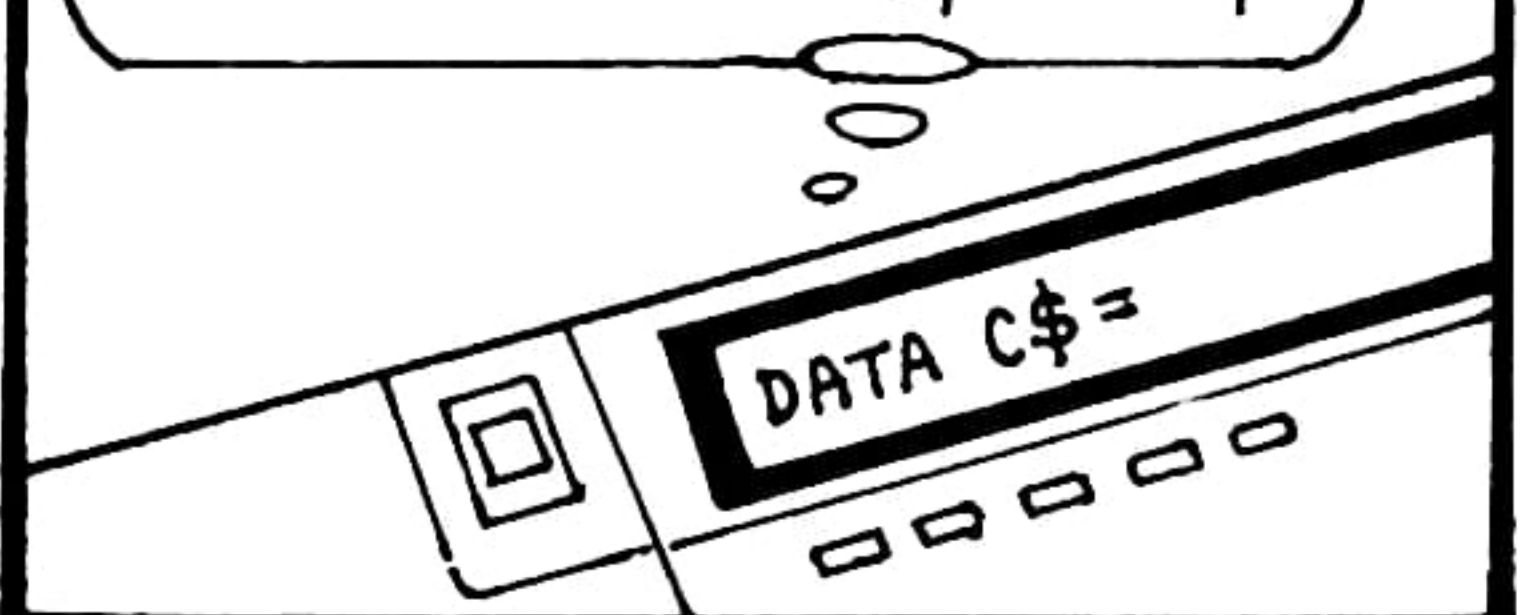
あとはN＝1のデータがくるまでこれをくり返します。

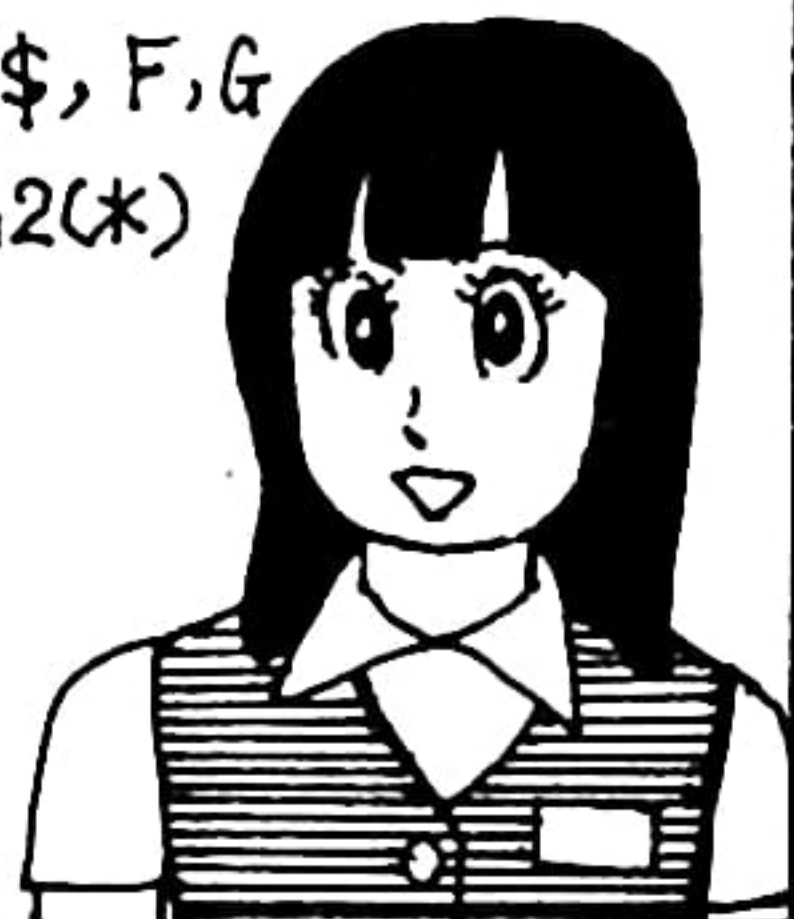
TAPE END

ポケコン側では文100に戻り次のデータを読みます。

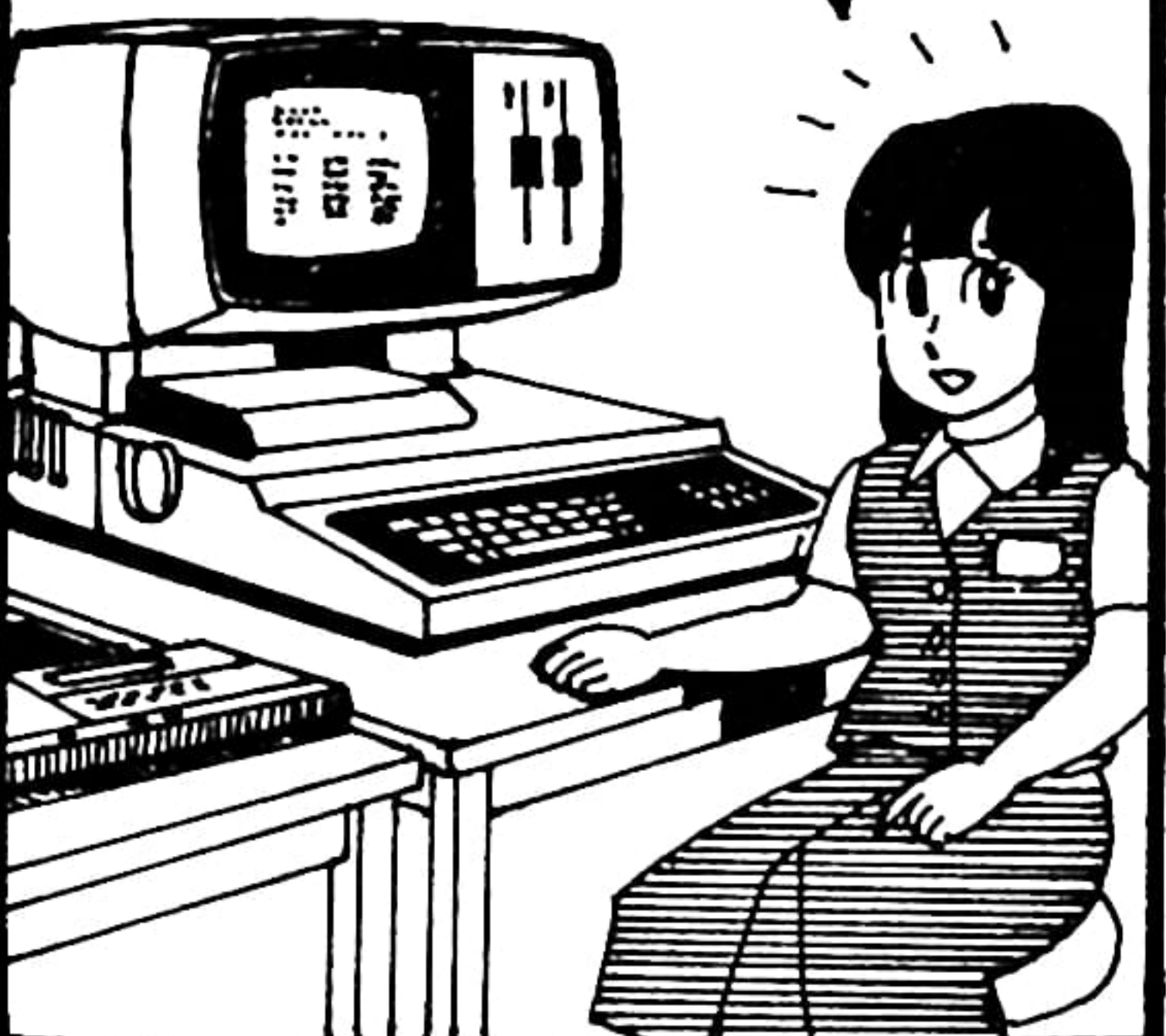
ピィーン ピィィィ

この表示は約2分間文250によって保たれます。文40に戻すとクリアされます。

変数にデータを入れていくプログラムはINPUT文を使って組んでください。テープレコーダーの録音(REC)スイッチをONにしてからプログラムをRUNします。

300: INPUT "DATA C$="; C$

DATA C$=

カセットへのデータの録音はINPUT #の代わりにPRINT #を使って行ないます。

```
PRINT#"D1"; C, D, E, N, B, R, S, U
PRINT#"D2"; C$, F, G
PRINT#"D3"; A2(*)
```

えっ、カセットへのデータはどうして入れるのか、ですって？

このプログラムで覚えておきたいのは文250と225でデータの訂正機能を持たせていることです。この場合99をインプットすることによってひとつ前に戻れます。この99はこのプログラムではあるはずのない数字なのです。

220: IF A2(I,J)<>99 THEN 230
225: I=I-2

参考のためにインプットしたデータをテープに録音するプログラムを作ってみました。1レコードのフォーマットは209ページのプログラムに合わせています。

```
 10:REM AN EXAMPLE
     OF TAPE INPUT
 20:Z=0
 30:WAIT :PRINT "T
    APE SET(REC)
    OK?"
 40:INFUT "C=";C
 50:INPUT "D=";D
 60:INPUT "E=";E
 70:INPUT "N=";N:
    REM TAPE END M
    ARK N=1
 80:INPUT "B=";B
 90:INPUT "R=";R
100:INPUT "S=";S
110:INPUT "U=";U
120:INPUT "F=";F
130:INPUT "G=";G
140:INPUT "C$=";C$
150:IF Z=1THEN 170
160:DIM A2(N,2):Z=
    1
170:WAIT 50
180:FOR J=0TO 2
190:FOR I=1TO N
200:PRINT "I=";
    USING "###";I;
    "    J=";J:BEEP
    1
210:INPUT "A2(I,J)
    =";A2(I,J)
220:IF A2(I,J)<>99
    THEN 230
225:I=I-2
230:NEXT I
240:NEXT J
245:PRINT "TAPE OU
    T"
250:PRINT #"D1";C,
    D,E,N,B,R,S,U
260:PRINT #"D2";C$
    ,F,G
270:PRINT #"D3";A2
    (*)
280:IF N<>1THEN 40
290:BEEP 5:PRINT "
    END":END
```

プログラムをRUNする前に新しいテープをセットします。またカウンタをゼロにします。PC-1500とテープレコーダを接続しリモートスイッチをONにしてから録音(REC)スイッチを入れます。テープは少し回しておきます。

90分テープ

文270のA2(*)は配列A2(I, J)全部の要素つまりN=10なら33個の変数を示します。特定の要素を抜き出して録音したい時は209ページのプログラムの文450のような工夫が必要です。

A(*)
I J 0 1 2
0 1 2 3 4 5 ⋮

次々に表示されるガイダンスに従いデータを入力します。配列へのインプットの時はBEEP文でピッと鳴るようになっています。

C = 20 ENTER
D = 18 〃
E = 13 〃
N = 10 〃
⋮
I = 5 J = 0
A2(I,J) = 13 ENT
ピッ
⋮

RUNすると、TAPE SET OK? と表示されます。OKであれば ENTER キーを押します。

離れた所にあるパソコンとつなぐ時は音響カプラと電話を使えば簡単にできるのです。

とにかく、使ってみること！
RS-232C

Nに1をインプットするとピッピッピッピッピと5回鳴ってプログラム終了です。これで例示のプログラムのインプット用テープができました。

N=1はあり得ない数字なのでエンド・マークとして使いました。

インプットが完了するとテープが回転を始めます。そして録音が完了するとテープは停止し、Nが1でない時は文40に戻ります。

ピュイ
イ〜ン
ピロピロ

RS-232CのQ&A

＊近ごろRS-232Cが注目を集めていますが…。
RS-232Cは、パソコン以前からテレックスなどの通信に使われていた規格です。大型コンピュータでは、端末機器との接続は常識になっています。パソコンもその性能が上がるにつれ、単独使用から端末機器と接続して使用することがパソコン活用の次のステップと考えられます。今やBASIC言語で簡単にデータの送受信が行なえるようになっています。あなたのやる気しだいで、すぐにもパソコンを使ったデータ通信が可能なのです。ただパソコンによって少しずつプログラムのしかたと入出力信号の方向が違います。でもそんなことは、次の４つのポイントさえおさえておけば大きな問題ではありません。

(1) 通信条件を一致させること
　通信速度
　パリティ
　データ長
　ストップビット

(2) コネクタピンの入出力方向の整合
　こちらの出力は相手の入力となり相手の出力はこちらの入力となる通常のパソコンはケーブルの中でクロスさせる。

(3) データ個数の一致
　送る側のデータ個数と受け取る側のデータ個数は一致していなければなりません。
　受信にエラーが起こるとデータがずれるので、必ずヘッダーを送り、受ける側はこれを確認してからデータを変数に読み込むようにします。

```
10␣OPEN␣"COM1:6N82"␣
   AS␣1
20␣INPUT␣#1,A$
```



```
30␣PRINT␣"ジュシン␣
   データ=";A$
```

↑　↑
この２つはKEYが違うので注意してください。

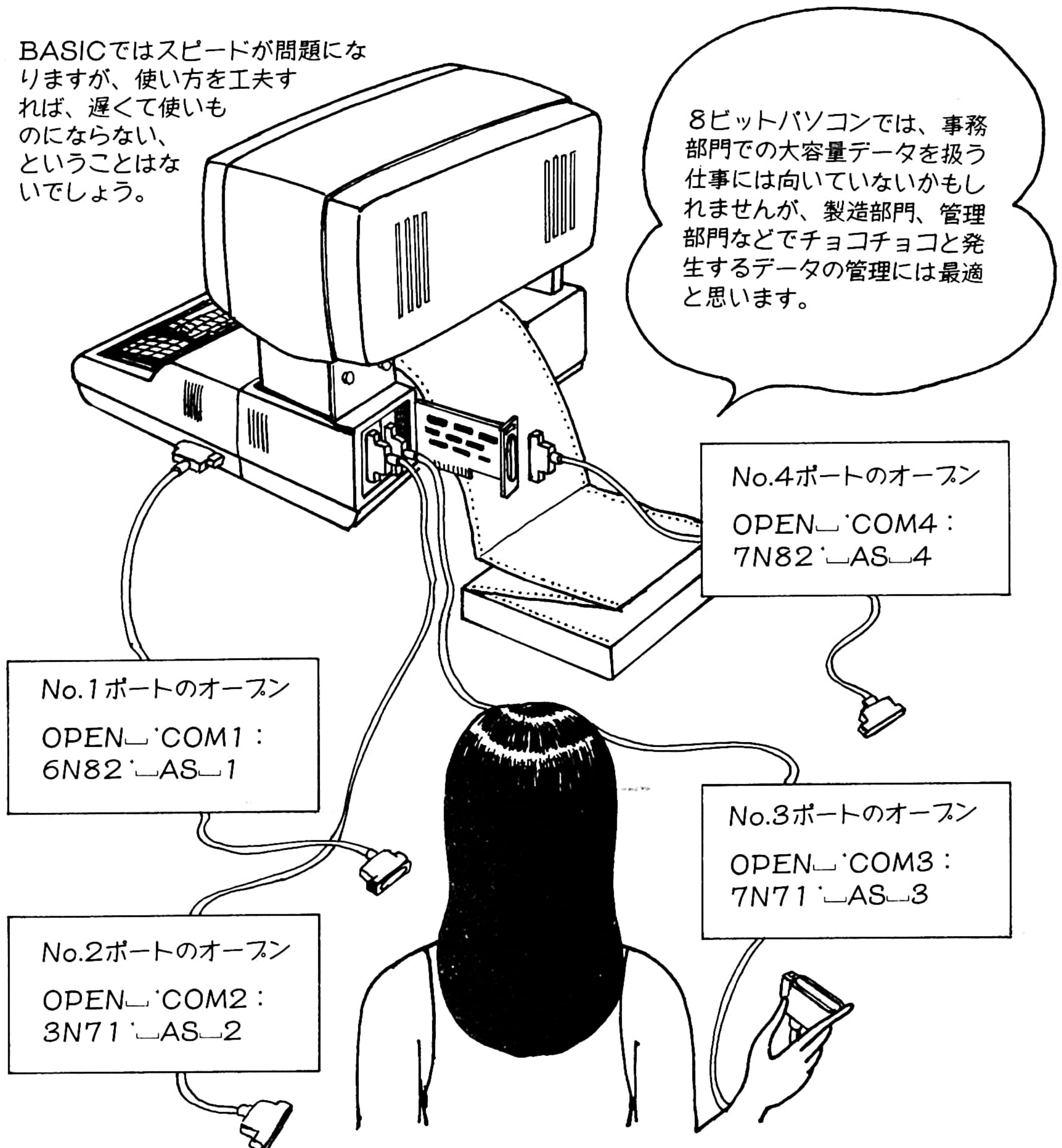
```
40␣GOTO␣20
```

(4) アスキーコードでの入出力
　数値は文字型にして送り、受信してから元に戻す。
　K$=STR$(K)
　↓
　K =VAL(K$)


```
10␣MAXFILES=2
20␣OPEN␣"COM:6N82"␣FOR␣OUTPUT␣#1
30␣OPEN␣"COM:"␣INPUT␣#2
40␣INPUT␣"ソウシン␣データ=";A$
50␣PRINT␣#1,A$
60␣GOTO␣40
```

＊BASICでの通信は実用的か？

パソコンが初めて世に出たころのBASICでは、RS-232C通信ポートを扱う命令はサポートされていませんでした。そこで、このポートを使うためにはマシン語プログラムをつくり、これと併用してBASICを使ったのです。このため、通信中はBASICの処理が止まったり、BASIC処理中は通信できないなどのことがありました。

現在では、BASICでこのRS-232Cが使えるようになったほかに、データの受信は割込みルーチンを使ってBASICの処理とは無関係に行なわれ、オープンしたバッファの中にデータが入ります。この図のパソコンでは最大４ポートのRS-232Cが使え、それぞれ独立したバッファにデータを受信できます。通信条件もそれぞれ別個に設定できます。

Ⓕデータ源から直接入力させる法
(ディジタル機器からの出力をBASICで扱うには)

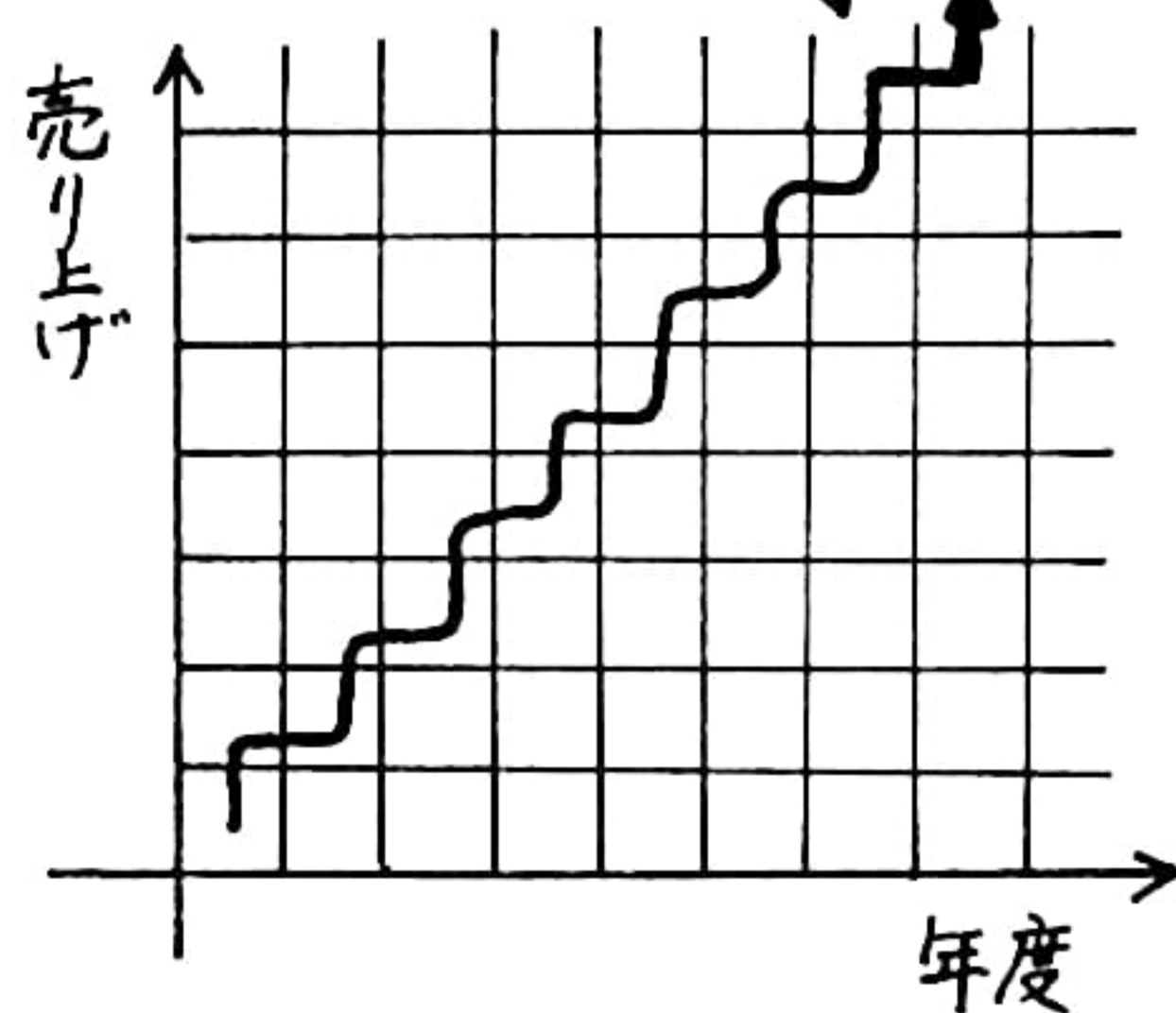
温度にしろ重量にしろ圧力にしろ、制御するためにはまず測定しなければなりません。これら測定器からは、ふつうBCD（2進化10進）の出力が出るようになっています。またバーコードリーダーなどの出力ポートはRS-232Cですから、これらを組合わせると多彩なパソコンデータ処理システムの構築が可能です。

ここでは、BASICでディジタル値をパソコンの変数に取り込むひとつの方法を、具体的な事例でお話します。ここで使ったパソコンはオールRAM型で2つのディスクドライブを持っていますが、他のどんな機種でも応用ができます。

RS-232Cにくらべるとハードがらみのことが多いので、誰でもすぐ応用が効くというわけにはいかないかもしれません。しかし、何でもそうですが全部自分でやる必要はないのです。ひとりでやれることなど知れていることも知っておく必要があります。

電算機発達の理由はこれらデータ処理のニーズがあったからです。

売り上げ

年度

官庁、学校、企業、商店などでは、たくさんのデータの整理、集計なども重要な業務のひとつとなっています。

なんとか損益一覧表

①工場のハカリとマイコンをつないでみたら

¥363,568.‥

395.0kg

95点

26H

300人

36.5秒

電算機やパソコン導入により事務の自動化いわゆるOAが急速に発達しています。

パソコンは電算機に比べると低価格であり、単能機として使っても問題にならないくらいです。

今、パソコンが出現し、その能力は電算機に近づきつつあります。

OS9

FUJITSU MICR 11

プログラム処理の都合上これらのデータはある程度まとめて処理されますが、これがバッチ処理です。

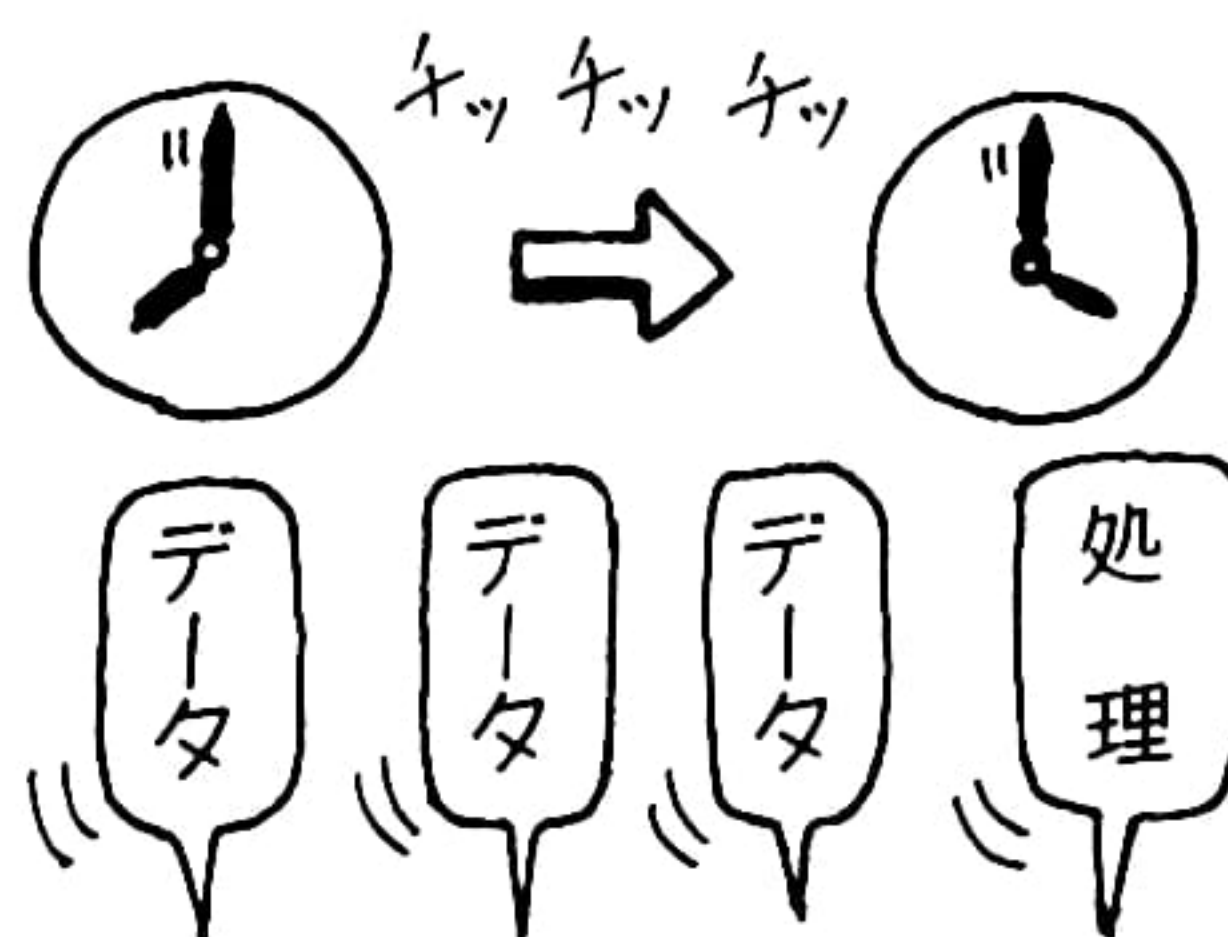
多くの場合データの発生からコンピュータへのインプットまでには時間的ズレがあります。

チッ チッ チッ

データ

データ

データ

処理

しかし、データの整理、集計などが自動化されてみると、今後はそのデータのインプット業務が目立つようになります。

この具体例が、あなたが今持っているニーズやこれからのOA革命対応へのヒントになればと願っています。

そうか!!

ガーン

工場の計量器(ハカリ)とパソコンをつないだ例から、なるほどこんな使い方もあるのだということを知っていただきたいと思います。

ハカリ

コンピュータとデータ源をダイレクトにつなげればOAの効果は増加するはずです。

データ源

?

やっていることがおもしろくなり知識の吸収に加速がついてくればそれまで難解に思えた理屈も自然にわかるようになります。ここではハカリとパソコンをBASICでつないだ具体例を書いていますがハカリのことを書いているのではないことはすぐにわかっていただけると思います。

私がいつも本当に知りたいと思うのは通り一遍のことでなく、具体的にどう扱えば動くのかということです。

データ

BASIC

CALL #1,B,P

自分の意志通り動かすことができてはじめておもしろいと感じると思います。

SORD I/O

この研究が重要

ここではパソコンはSORD M200を使っています。

計重サイクル

① コークスチャージ総重量メモリ

② コークス排出後残重量メモリ

③ 正味投入量を計算する

ハカリ

コークス炉断面

この計重システムはコークス炉のうえに置かれ、コークス炉へのコークス投入量を管理するためにつくられたものです。

ホッパーゲート開閉シーケンス（装炭車）

大型指示計

888.8 t

電源AC100V

はかり制御盤

はかり

電源AC100V

電源AC100V

設計された機器の構成はこのようになっています。

ロードセル
ハカリ
ディジタル指示計
プリンター
大型ディジタル指示計
888.8 t
炉番
データプール
定量
計重システムのハードウェアの構成はだいたいこんな感じです。
CPU Z80 ROM RAM
I/O カード
ROM RAM
I/O カード
I/O カード
Z80占有バス
I/O カード
I/O カード
I/O カード
リレー回路
端子台
外部信号取合端子台
CPUへ

台車はハカリを出て決められた炉のNo.の穴にコークスを払い出します。

コークスをチャージし、定量まで入れます。そして安定した時のデータを取り込んでこれを総重量Wtとします。

123.6

ホッパーの下に装炭車が入り、コークス投入のボタンを押します。ホッパーのゲートが開きます。

はかり
85.4

この時が普通のパソコンへのデータ発信へのタイミングとなります。

① 炉番 3桁
② 総重量 4桁
③ 残重量 4桁
④ 正味重量 4桁

②と③から④は計算できるが照合用として発信する。

この時、炉に払い出したコークスの正味重量が計算できます。(Wn)

$W_n = W_t - W_o$

W_t
W_o

そして装炭車は再びハカリに戻りそのままで計重します。この時のデータを残量Woとして取り込みます。

はかり
82.2

プール数表示が0でない時は、パソコン側でデータを受け取っていないことになります。

早くしてね。だいぶたまったよ。

このプール数を目視確認できるようにこれを表示するようにしました。

プール数

88

No.	炉番	総重	残重

パソコン側の要望は、別のプログラムを走らせている間に発生したデータはハカリ側でプールしてほしいということでした。

マルチプログラミングはできないんですよ。

集計プログラム

計重プログラム

送信データ

炉番3桁，総重4桁，残重4桁
正味4桁

コード

BCD，オープンコレクタ，負論理，並列

制御信号

ストローブ
アクノリッジ
タイミング
SORD側 I/Oの仕様による

ケーブル

0.25SQ×50C ツイストペア，5m

ハカリ側のデータの仕様は次のように決めました。

これを簡単に説明します。

パソコンがデータを受け取ると、プール表示がゼロになるまでためたデータを出力します。

データ送っていいぜ

トトンツ

……3210

10進 1 2 3 6
BCD 0001，0010，0011，0110

0 8 2 2
0000，1000，0010，0010

このコードは10進数を2進化コードにして順に並べたものです。

人間側に有利なコードです。

	BCDコード			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

コード表

BCDコードは2進化10進コードのことです。4ビットで10進1桁を表わします。10以上のA、B、C、D、E、Fのコードは含まれません。

ABC DE F?

たとえば表示器へのデータ入力用コネクタを抜いた時負論理だったら0を表示し、正論理だったら888と全桁フルコードを表示します。

負 0

正 88888

負論理というのはコード表においてSINK時の信号を'1'にするものです。

SINK

VCC 5V

①

0V

オープンコレクタとはトランジスタのコレクト側を負荷を接続しないで端子に出すことです。自分側のノイズ対策に効果があります。

VCC

ツイストペア

オープンコレクタ

相手側

5m

自分側（ハカリ）

ここにI/O基板を入れ、ハカリ側からのデータを受け取れるようにするのです。

一番下にすでに基板が一枚入っています。そして、その上にさらに3枚分の基板を入れるスペースがあります。

スロット3
スロット2
スロット1

次はパソコン側です。裏側のフタをはずしてみます。

CPU バス

I/O基板一枚でアウトプット32ビットインプット32ビットの入出力ができます。

INPUT
コネクタ
OUTPUT
コネクタ
8bit ×4ポート
8bit ×4ポート

これを使うためには前に紹介したI/Oマニュアルを読まなければなりません。

PART：2
HC-DIO
M200用デイジタル入出力ユニット
ハードウェア、ソフトウェア 説明書

データはパソコンにとってはとってはすべてインプットになるため、アウトプット側は使いません。

IN
OUT
こっちだけを考える

ハード的にはこのI/O基板が2枚必要になります。

ハカリ側からのデータは全部で60ビットになります。BCDコードで15桁分です。これを受け取るためには、

炉番 3桁
総重 4桁
残重 4桁
正味 4桁
計 15桁
15桁×4bit
=60bit

基板とHC-10BPは専用コネクタで接続します。

25ピンコネクタ
50ピンコネクタ
CN1-1
CN1-2
CN2-1
CN2-2
メーカーから有償で提供されます。

データ取出しのためにコネクタが必要です。そのコネクタ取付パネルとして用意されたのがこのHC-10BPです。

基板0と1の区別は基板についているディップスイッチで設定します。

このスイッチは4つ、つまり4ビットあるわけで、0～Fまで16通り設定ができます。

SW0
1 2 3 4
OFF
0
1
3 2 1 0

炉番 下2桁 → ポート0
上1桁 → 1
総重 下2桁 → 2
上2桁 → 3

0

残重 下2桁 → ポート0
上2桁 → 1
正味 下2桁 → 2
上2桁 → 3

1

次はハカリからのデータをそれぞれの入力ポートに割り当てていかねばなりません。

ポートとデータのビットの関係はマニュアルのコネクタ表に載っていますから、それを見て相互接続します。

たとえば総重の上2桁は基板0のポート3に入るのでデバイスNo.は 03と表現されます。

この基板のNo.を上桁にポートNo.を下桁にして、デバイスNo.にします。

総重

デバイスNO.
基板 ポート
0 3

これでソフト的にも0か1で基板の区別ができます。

0001

0000

1

0

次に、この中のひとつのポート の制御方法について調べてみましょう。

1ポートは8bit単位の入力でほかに制御信号が2つついています。

8bitデータ

ポート

DEV. NO.

		DEV. NO.
炉番	下2桁	00
	上1桁	01
総重	下2桁	02
	上2桁	03
残重	下2桁	10
	上2桁	11
正味	下2桁	12
	上2桁	13

デバイスNo.はこの8つに決まりました。I/Oプログラムではこのデバイス No.を参照します。

データ

INPUT PORT

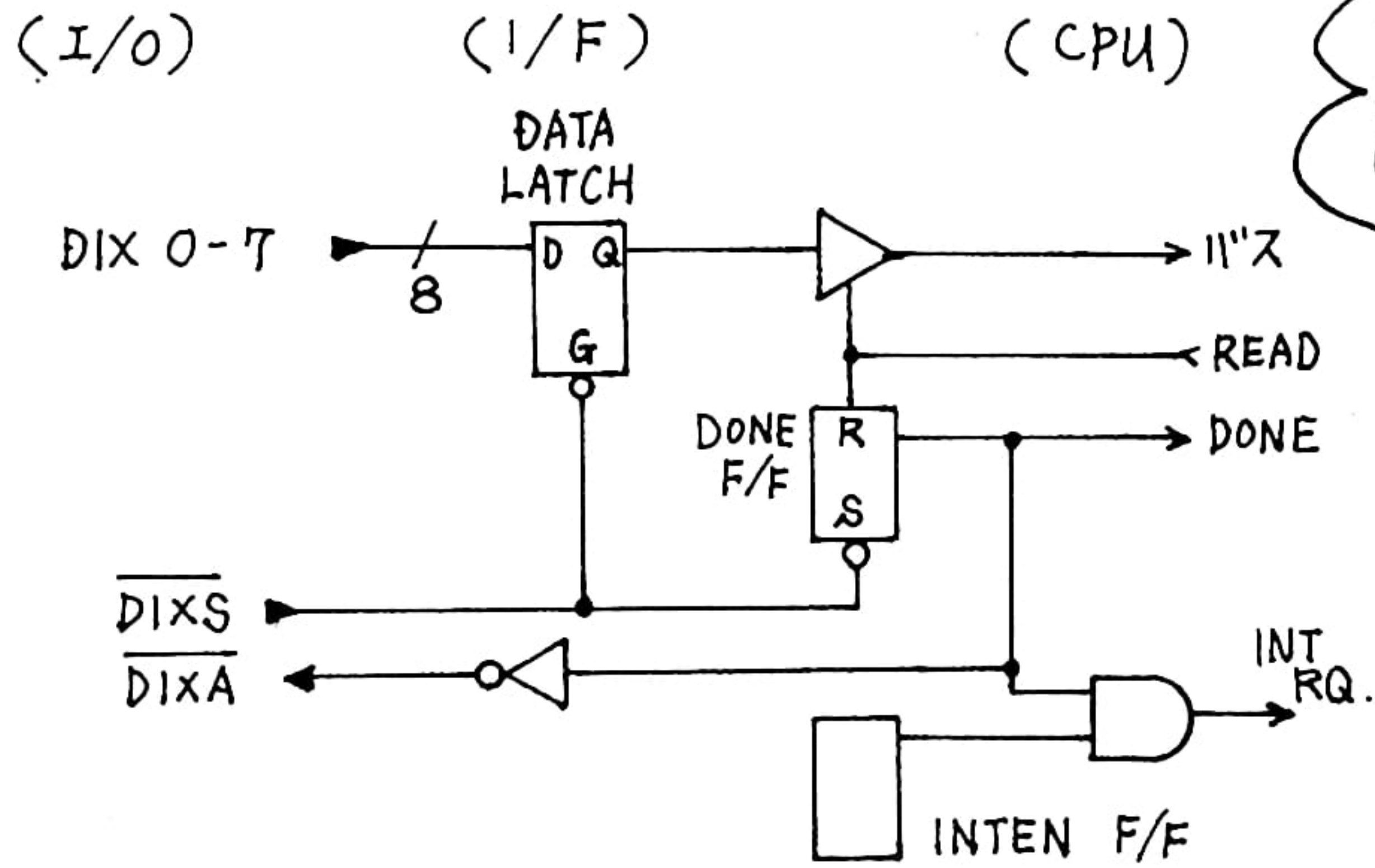
ALレジスタ

03

メモリの時のアドレスと考えてください。

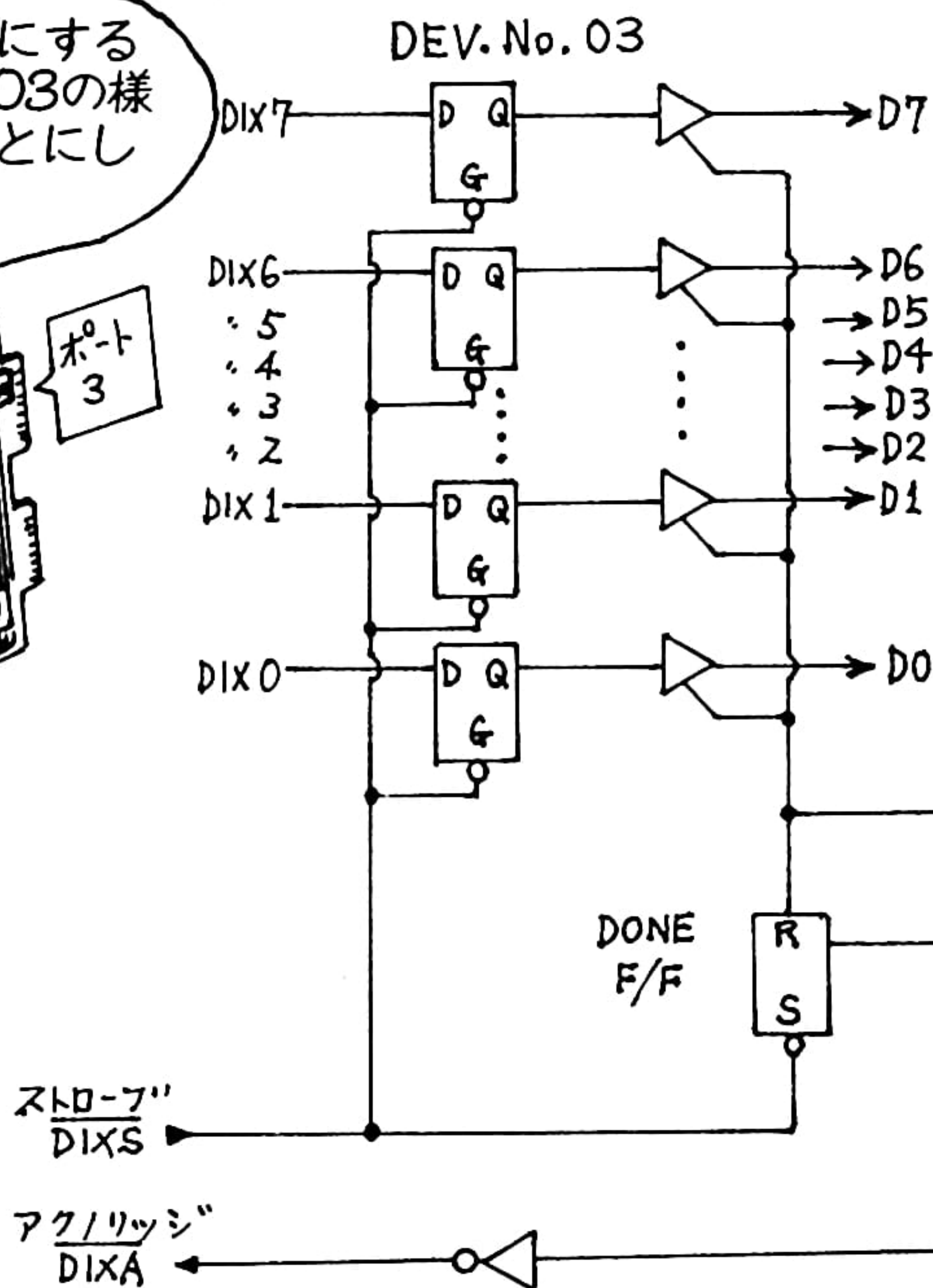
IN命令で操作できます。

ハードのことが全然わからない人にはこの図の解読はむずかしいと思います。

(I/O)
(I/F)
(CPU)
DATA LATCH
DIX 0-7
8
D Q
G
バス
READ
DONE F/F
R
S
DONE
DIXS
DIXA
INT RQ.
INTEN F/F

マニュアルの説明図をそのまま転記しました。

具体的な内容にするためにDEV.No.03の様子を見てみることにします。

ポート3

DEV.No.03
DIX7
DIX6
5
4
3
2
DIX1
DIX0
D Q
G
D7
D6
D5
D4
D3
D2
D1
D0
READ
DONE F/F
R
S
DONE
ストローブ DIXS
アクノリッジ DIXA

なんか無視されてる感じがするけど僕抜きでは考えられない。

僕Aレジスタ

そこでこのマンガ風に書き直してみました。

IN命令実行時のINP信号と考えられる

4番目のビットに'1'書き込み

7 6 5 4 3 2 1 0

Aレジスタへ

アルバイト

DEV.No.
04

ソフト的には、CALL文実行時このレジスタを参照している。

Aレジスタから

I/O命令下のレジスタとしてDEV.No.OBで書き込みできます。

INTEN F/F

7 6 5 4 3 2 1 0

インタラプトリクエスト

この割込み処理ルーチンを活用するためにはCPU Z80の知識が必要です。

INT

データのチェンジを示します。

データ 28 31 29 30
ストローブ
アック
ラッチ 31
READ

CPUのAレジスタには31が入る

インタラプトイネーブルはソフトでセットします。ただし、機械語レベル。

アルバイト

DEV.No.
OB

その手法についてはアセンブリ言語マニュアルに書いてありますけどちょっとわかりません。これは不親切というわけじゃなくて程度が高いというわけです。
SORD
そして、アセンブラで作成したリロケータブルプログラムとBASICとを結合させるCALL文の作成など高度なソフト上のテクニックが必要になります。
データ入力を割込みプログラムで処理する場合は、アセンブラ言語でのプログラムが必要になります。
INTEN F/F
アルバイト
OB
このDONE F/Fは8ビットのI/Oレジスタとして働き、それぞれのビットにはそれぞれ意味があります。
アウトプットポート
インプットポート
7 6 5 4 3 2 1 0
このプログラムで使用するのは、ハカリ側からくるストローブ信号を受け取るDONE F/F(フリップフロップ)です。
アルバイト
R
S
DONE
DIXS
そこで、ここではメーカーのSORD側で用意しているI/Oプログラムをそのまま使うことにしました。
DONE F/F
アルバイト
DONE F/Fの内容
(ダン・フリップフロップ)
3 2 1 0
3 2 1 0
このハカリのシステムでは、04と14のレジスタとして参照されるはずです。
I/O
04
14
この番地はIN命令で指定される。
このI/Oレジスタは下桁が4で上桁は基板のスイッチの設定値で決まります。
SWO
4
DONE F/F
指定席
DONE F/Fに1が立ったからといってパソコンにデータが読み込まれるわけではありません。パソコン側でデータを受け取るプログラムをBASIC言語で組んでおく必要があります。
あせんなって。とりあえずDIXAを送っとく。
ハカリ
早くしてくれよ。
ハカリ側からDIXSを送ると、送られたデバイスNo.のDONE F/Fに1がたちます。
1
DONE F/F
DIXS

ではいよいよプログラムです。BASIC言語で外部からデータをインプットするんですよ。

②データを取り込むためのプログラム

これがその文の書き方です。

これだけではなんのことかわかりませんので、ハカリのデータインプットを例にしてご説明します。

CALL #n, argument1, argument 2

まず最初の#n これは、サブルーチンの番号です。全部で4つあります。

#0…バイナリー入力
#1…BCD入力
#2…バイナリー出力
#3…BCD出力

#1と書くと、BCDコードを使った入力サブルーチンに処理がわたされるわけです。

CALL#1, B, P

BASIC インタプリタ	#0 バイナリー 入力ルーチン	#1 BCD 入力ルーチン	#2 バイナリー 出力ルーチン	#3 BCD 出力ルーチン

BASICに付加したI/Oプログラム

argument1 これは入力値を入れる変数名を書きます。たとえば、これをBとすると、

B

BASICのプログラムはこの変数Bでインプットデータを扱えるのです。

```
80 CALL#1,B,P
90 PRINT"B=";B
```

B=56

01010110

BCD2桁

ハカリ

argument 2 これはパラメータの設定です。一次元配列を使ってプログラムします。

DIM P(10)

このパラメータはCALL文実行の前にそれぞれの要素に必要な数値をセットしておきます。

P(0)… エラーの内容が入る
P(1)… デバイスNo. 基板の0か1
P(2)… ポートNo.に1加えたもの(1,2,3,4)
P(3)… データ数(この場合1)
P(4)… 正論理/負論理(この場合1)
P(5)… ストローブ/レベル(この場合0)
P(6)… インターバル(この場合必要なし)
P(7)… 自動開始/EXTACK開始 (この場合必要なし)

データはひとまず、配列Aに読み込むことにします。

デバイスNo.　ハカリ

		デバイスNo.	ハカリ	
A(0)	←	0	0 ←	炉番
A(1)	←	0	1 ←	炉番
A(2)	←	0	2 ←	総重
A(3)	←	0	3 ←	総重
A(4)	←	1	0 ←	残重
A(5)	←	1	1 ←	残重
A(6)	←	1	2 ←	正味
A(7)	←	1	3 ←	正味

これで準備完了。では、いよいよプログラムにとりかかりましょう。

エラーなく実行されたらこのように表示されるはずです。

```
A(0)= XX
A(1)= XX
A(2)= XX
A(3)= XX
A(4)= XX
A(5)= XX
A(6)= XX
A(7)= XX
```

```
10:DIM A(7),P(10):B=0
20:P(3)=1:P(4)=1:P(5)=0
30:P(1)=0
40:FOR I=0 TO 3
50:P(2)=I+1 :CALL #1,B,P
60:A(I)=B :PRINT "A(";"I=";B
70:NEXT B
80:P(1)=1
90:FOR I=0 TO 3
100:P(2)=I+1 :CALL #1,B,P
110:A(I+3)=B:PRINT "A(";I+3;")=";B
120:NEXT B
130:GOTO 30
```

ホントに読んでいるかどうか確かめるためのものなのでこんなのでいいのではないでしょうか。

あとはBASICでデータ処理できますね。

例 炉番123、総重158.4t、残重110.6t、正味47.8t

```
A(0)= 23
A(1)=  1
A(2)= 84
A(3)= 15
A(4)= 06
A(5)= 11
A(6)= 78
A(7)=  4
```

DONE F/F

P(1)=0 → 7 6 5 4 3 2 1 0: □□□□1111

P(1)=1 → 7 6 5 4 3 2 1 0: □□□□1111

CALL #1, B, P

RETURN

I/O BCD入力ルーチン

8回のソフトスキャン

	7 6 5 4 3 2 1 0	
00	0 0 1 0 0 0 1 1	炉番下2桁 (23)
01	0 0 0 0 0 0 0 1	炉番上2桁 (01)
02	1 0 0 0 0 1 0 0	総重下2桁 (84)
03	0 0 0 1 0 1 0 1	総重上2桁 (15)
10	0 0 0 0 0 1 1 0	残重下2桁 (06)
11	0 0 0 1 0 0 0 1	残重上2桁 (11)
12	0 1 1 1 1 0 0 0	正味下2桁 (78)
13	0 0 0 0 0 1 0 0	正味上2桁 (04)

まだかんじんなこと、やってなかった。
まだまだ動かないもんね。
やったー
完成と
思ったけど
きちんとしたデータにしたい時はたとえば総重の場合このように計算するといいでしょう。
C=(A(3)*100+A(2))/10
検算
(15×100+84)÷10
=158.4
特殊なコマンドを使って簡単にできます。
コマンドをインプット
BASIC・SAV
DIOE・RB
XBASIC・SAV
OS
BASIC
DIOE・RB
DIOE
XBASIC
RLDR・SAV
I/Oプログラムをくっつけてない普通のBASICではこのプログラムは走りません。その I/O付BASICインタプリタは、
システムプログラム付
コマンドは、えーと Ready RLDR1:XBASIC/S/C␣BAC.LO/X〔␣1:TRANSA␣BA‥‥␣DR RETURN
SORDのOSが必要なI/Oルーチンをつないでくれるんです。
新しいXBASICは元のBASICよりも約1.7Kバイト大きくなります。
BASIC+1.7Kバイト
XBASIC
ワイドだよ。
確かにXBASIC.SAVという新しいプログラムが作られています。
XBASIC・SAV
あっ、OSに戻った。ではファイルを調べてみましょう。
Ready
たったこれだけで新しいI/O付XBASICをつくってくれるんだもんね。すごいというのが実感です。
カチャ
カチャ

どんなプログラムでも一度は誰かがインプットしなければなりません。

XBASICがメインメモリにロードされました。これでBASIC I/Oプログラムがインプットできます。

カチャ

Ready
XBASIC
*

これでもうできたも同然。OSに戻して XBASIC RETURN

Ready
XBASIC

XBASIC
RETURN
ポン ポン

プログラムはDEV.NO.00からの入力待ちとなっていますので、次はハカリの方からデータを送ります。

インタプリタモードに戻してRUNコマンドを入力します。

*
RUN

インプットが終わったら

AUTO 10
10: DIM A(7), P(10
20: P(3)=1: P(4)=1:
30: P(1)=0
40: FOR I=0 TO 3
50: P(2)=I+1: CALL
60: A(I)=B

カタ カタ カタ

ハカリ側から８つのポートに00、11、22・・・・・・、88、99と順にデータを送るしかけです。

このボタンを押すと、

TEST

ハカリの方にはI/Oテスト用プログラムをつけてあります。

接続長さ5mは限界です。これ以上だとデータの保証はできません。

ソフトだけでなく配線のチェックも兼ねることになります。

パソコン側では順に表示されるのでこれを見ていれば正誤のチェックができます。

A(0)=33
A(1)=33
A(2)=33
A(3)=33
A(4)=33
A(5)=33
A(6)=33

SORDのI/O
マニュアルとアセ
ンブリ言語マニュ
アルで一応ここま
でやりました。

とにかくハカリからのデータをBA
SICレベルのデータとして読み込
むことに成功したのです。

この例では2桁ごと8つのポー
ト(桁パラレル)でデータ転送し
ましたがハードを減らす方法も
あります。

CALL #1.B.P

2桁ごと1つのポートに8
回データ転送する方法で
す。桁シリアルで両側の
ソフト変更ででき
ます。

こうするとハード的には1
つのポートでいいのです
から基板は1枚、ケーブ
ルも最低10Cのツイスト
ペアがあればいいの
です。

CALL#1.B.P

1計重でBCDコードが2桁ひ
とつのポートに8回入る。BA
SIC I/Oプログラムはこのよ
うになるでしょうか。

CALL

データ

DIXS

```
P(1)=0:P(2)=0:B=0
FOR I=0TO7:CALL#1,B,P:A(I)=B:NEXT I
```

パラかシリかど
っちがいいのでし
ょうか。

コスト

桁シリだったらチ
ェックがむずかしい
だろーな。

I/Oプログラムは、一応FDに
SAVEしておきましょう。

カチャ

メーカーの用意したマニュアルだ
けでなんとかここまでやりました。
でも、自由自在に使いこなしていく
にはそれなりの能力と経験が必要
だと思います。

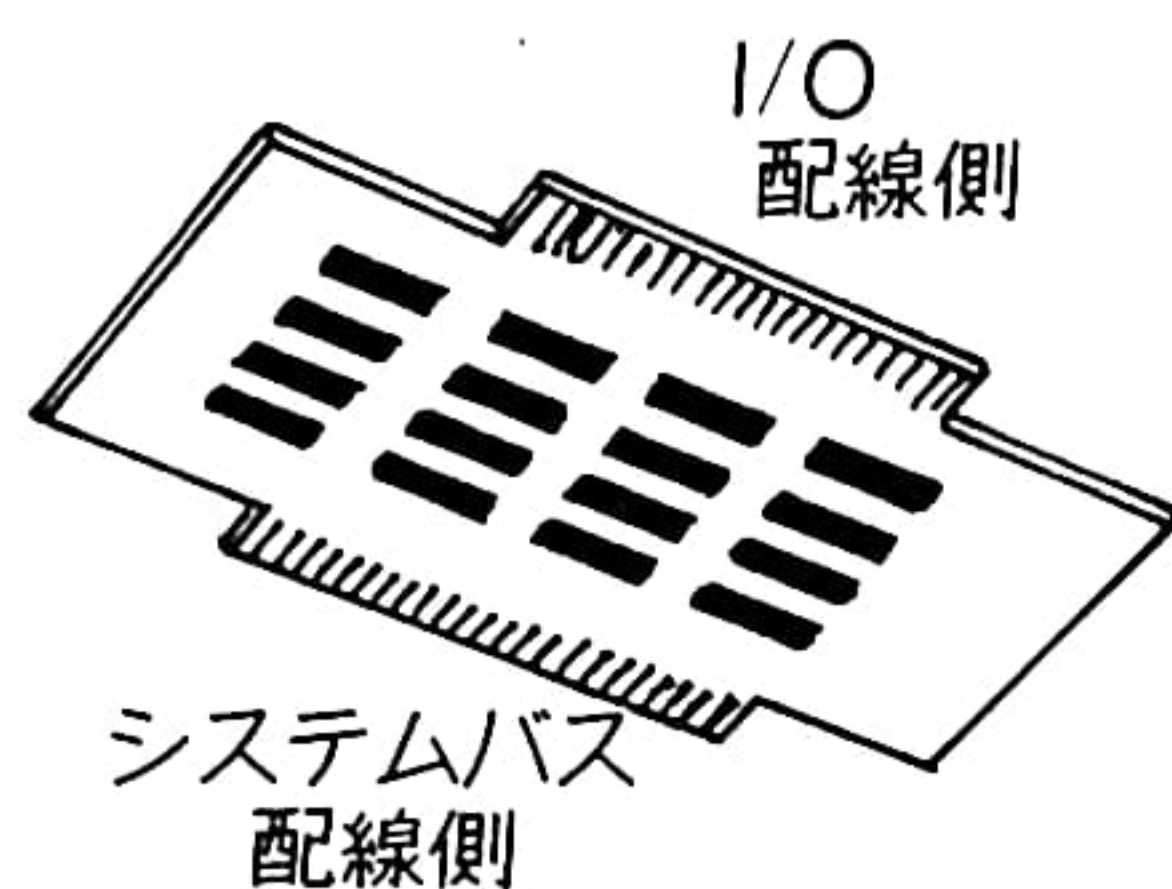
ブツブツ

結局はCPU
のことがわ
かってないと
ダメなのよ。

こうすると制御盤内のラックに組み込んだ時、システムが単純化できるのです。

I/O基板にはシステムバス用コネクタの反対側にI/Oのための配線用コネクタがあるものが便利です。

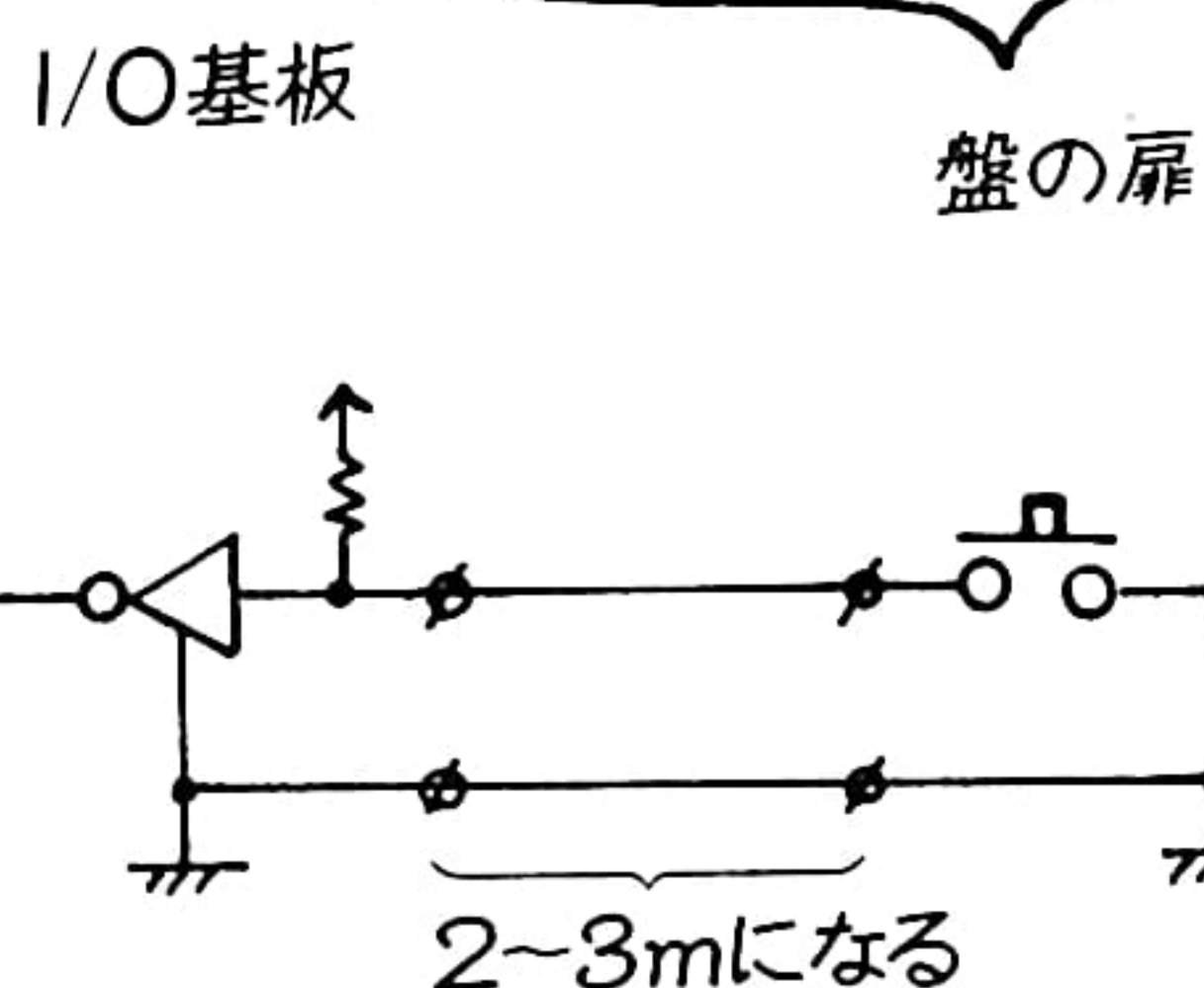
I/O
配線側

システムバス
配線側

③ I/O回路のノイズ対策

これはボタンによるインプットの回路例ですが、この図面では現場でどう配線されるかわかりません。

I/O基板

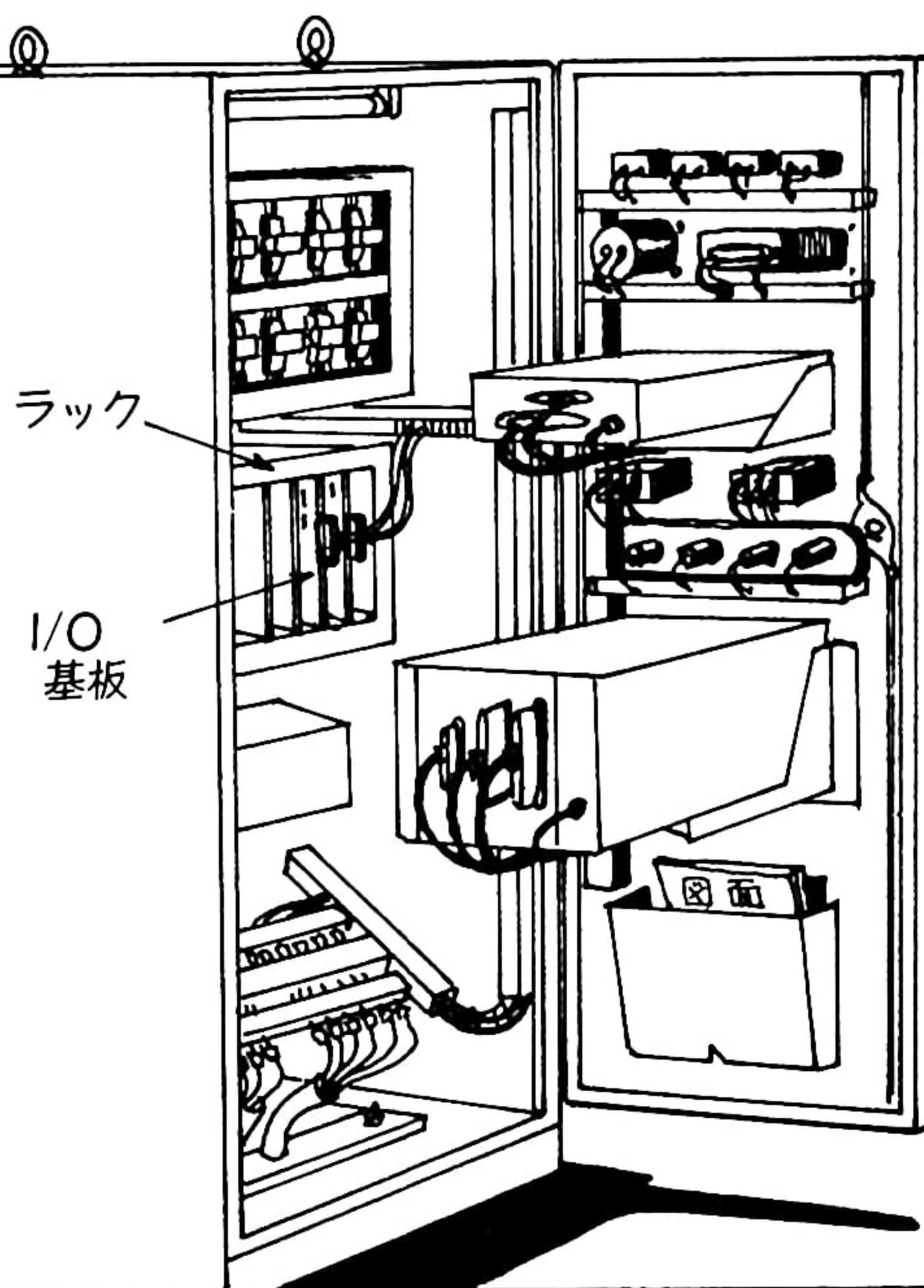
盤の扉

2〜3mになる

つまり、応答速度やノイズへの対策が必要になってくるのです。

ここで実用上どんな問題があるかといいますと、I/O基板から出た信号線はほとんど盤の正面のボタンや表示器に出て行くため配線が長くなるということです。

ラック

I/O
基板

図面

図面にはこのようにツイストペア線を用いることを明示し、コモン側はI/O基板のついているラックのアースまで引っ張ってくる必要があります。

ツイストペア線で
配線することを示す

ヒーッ

AC100V

ノイズテストにはノイズシミュレータが使われます。電源ラインに500Vとか1000Vのサージ電圧を発生させる装置です。ノイズ対策のない回路の場合はひとたまりもなく誤動作します。

このツイストペア線を用いるとコモンモードノイズに対してかなりの効果が期待できます。効果はケースバイケースであんまり効かない場合もあります。

この電圧ラインに発生したノイズはIC回路の定電圧装置(AVR)では吸収しきれずすべてIC回路へ入っていきます。

表示器
プリンタ
スイッチ
I/Oポート
IC回路
AVR
ノイズシミュレータ
AVR（エーブイアール）
定電圧装置

ノイズシミュレータを経由した電源には図のようなサージ電圧が発生しています。

500 〜 1000V

まず、信号線のツイストペア化の他に、電源ラインについてもツイストすることによって同一ダクトに収納される信号線への影響を少なくするようにします。

XO
YO
GND

フリップフロップの反転やCPUの暴走と思われる現象が起こったら、

したがって、このノイズシミュレータを使ってテストしておけば電源ラインのノイズ対策ができるわけです。

500Vぐらいでは平気だよ。

またこれらのノイズ対策機器のアースはきちっと接地されていなければ効果はありません。

GND

さらにシールド付トランスをつけるといっそう効果的ですがコスト高となります。

TR
X
Y
100V
100V
X1
Y1
GND

また、盤に入ってすぐの位置にノイズフィルタをつけるのも効果があります。

NF
X1
Y1
XO
YO
GND

設置したあとで単発的に誤動作し、設備担当者やメーカーを泣かせるということもあるのです。

ウィィ〜ン

I/Oにはノイズ対策をお忘れなく

メーカー側では納入前にこのようなチェックを行なうのですがノイズ環境は設置場所によりそのつど違うというのが実情です。

チェックはプリンタなどで一定時間間隔でプリントさせ、エージングを兼ねて行ないます。

パチャ
パチャ

ここでI/O関係のハードウェアのお話もちょっとしておきましょう。

パイロットランプ
ブザー
モーター

マイコンやその他のICでは大きな電流を必要とする負荷、たとえばモーターなどのスイッチングは直接にはできません。

パッ
DC24V 90mA
ブーッ
AC100V 15mA
AC100V 0.5A

このスイッチングに欠かせないのがリレーです。

早く僕を紹介してよ。

おれのマンガだぞ

リレーの原理はコイルに電流を流して磁石にし、その磁力で接点(スイッチ)を入れたり切ったりするものです。

a
b
c
接点
コイル

図面ではこのように書きます。ひとつのリレーにはいくつかの接点がついていますので必要なものを選択できます。

a
b
c
a
b
c
コイル

これはa接点のシンボルです。コイルに電流が流れた時にくっつきます。くっつくのをメークすると言っています。

これはb接点です。コイルに電流が流れた時にはなれます。はなれるのをブレークするといいます。

b接点を黒く塗るのは直観的にa接点と区別するため

これはC接点、a接点とb接点の組合せです。略してa接、b接と言う方が多いですね。

a
b
c

小さなものから順次大きなものに伝達して目的の負荷を開閉するのを接点増幅といいます。

これがホントのリレーだね

モーター
パワーリレー
ミニチュアリレー
リードリレー
ICドライバー
CPU

大小いろんな形のリレーがあります。

そして、この接点に流せる電流の大きさによって

電気的にはこのように信号が伝達されます。

X1 R2 R1 M1 I X1 R2 IC ドライバー ラッチ OUTPUT ポートの 1ビット OUT

リレーの記号はコイルの中に書きます。

リレー回路がある時はＩＣ回路へのノイズ対策が必要になります。

パチッ パチッ

リレーのコイルについているダイオードは直流回路でのサージ・キラーの役目をします。

バイパスダイオード

OFF時のスパーク

パチッ

ＩＣドライバーの出力はオープンコレクタです。でもGND(グランド)側はリレー回路と結んでおかなくてはなりません。

IC ドライバー I 信号

AC(交流)回路ではダイオードは使えません。コンデンサと抵抗を組み合わせたものを使います。

R C

既設リレー盤のある所にＩＣ盤を設置する時は、それらのリレーにサージキラーをつけなければならない時もあります。

光が出たところで光ファイバーのことについてちょっと考えてみたいと思います。

あれ、僕の話はもう終わりなの？

フォトカプラは、光で信号を伝えるため、電気的には分離されることになります。

また、ＩＣ回路とリレー回路を完全に分離したい時には、電源を個別に用意して、フォトカプラを使います。

V1 V2 I I 信号 フォトカプラ

光ファイバーの原理は２つの屈折率の違うガラス管で構成された２層パイプです。光は内側にのみ屈折して進むため遠くまで、ファイバーが曲っていても、届くのです。

光ファイバーというのは、今はやりの最先端技術のひとつですからよくご存知と思いますけど…

現在使われているメタルケーブルには伝送速度に物理的な限界があります。

それは光のスイッチング性と対ノイズ性がメタルにくらべてだんぜんいいからです。

メタル
光ファイバー
電流
光
とん
ツーツー
とんツー

たったこれだけのものなのにどうして未来産業とかいわれ、注目されているのでしょう？

光通信

この２つは入力信号に対して出力信号をなまらせてしまうように働きます。

P1での波形
P2での波形

つまりケーブルが長ければ長いほどメタルの内部抵抗と対地間の浮遊キャパシタンス容量が増えます。

受信
発信
P1
P2
P3
r
c
c
メタル ケーブル

また、0か1かどちらとも保証できない入力電圧の範囲があってこれがスレッシュホルドレベルと言われるものです。

ICも、ビットが1は5V、0は0Vというわけではなく、ある範囲を持って1か0かに分かれます。

P2での電圧
5V
スレッシュホルド
0V
P3での信号
0
1
0
1
不定
不定
不定

受信側ではこのなまった部分では正しい情報を受け取れませんからパルスの幅をある程度長くして安全サイドの設計をする必要があるのです。

信号
t
読込みタイミング

重要な信号線はシールドする必要があります。

という具合に途中で外部の磁界や電界の影響を受けやすいということです。

メタル通信にはもうひとつ決定的な弱点があります。それは…あら、いい男…

ノイズ

3つのパルスになっており発信内容とは違った情報が伝達されることになる。

たとえば電子化の進んだ都市が核攻撃を受けた場合など、

某国、電子部品老朽からミサイルを誤射…。

核爆発による強力な電磁波により、

ピカッ

通信回線のメタルに強力なサージ電圧が発生して、それにつながっているコンピュータをはじめとする電子機器がパンクする危険があります。

バチ

バチバチ

直接破壊されなくても、情報機能の停止等により都市は大きなダメージを受けます。

ドドーン

この点、光通信は光ったかどうかがビット情報となるためファイバーが破壊されない限りOKなわけです。

ピカ

うん 届いてる

ピカ

ファイバーの両端は発光ダイオードと受光ダイオードで対になっており、ファイバー自体が長い長いフォトカプラと言えます。2つの回路を電気的に分離することにもなります。

発信パルスと同じであり情報が正しく伝達される。

こんなわけで、光ファイバーは注目を集めているのです。

ファイバー

メタル

我々メインフレームはより高速処理が可能なように工夫されるでしょう。
マルチプログラミング
タイムシェアリング
32ビット
ウルトラスーパーコンピュータ
CPUバス
僕はチャネル。CPUのI/O待ちを減らします。
I/O情報と
I/Oデータ
チャネルバス
16ビット
プリンタコントローラ
8ビット
プリンタ
ディスクコントローラ
ディスクドライブ
テープコントローラ
テープドライブ
パンチカードコントローラ
リーダーパンチャー
情報処理技術者
衛星通信システム
画像情報システム（キャプテンなど）
テレ・メータシステム
電話交換システム
電話帳
これからはより中央集中化、分散化、大衆化へと進んでいくでしょう。
新しい家庭生活の予感がするわ。
デジタル化と光通信化が進む
電力計
ガスメーター
防犯センサー
テレビ
プリンタ
受信ユニット
テレ・メータユニット
電話交換システム
制御用マイコンの場合は8ビットで十分なことが多いですね。
普通の業務なら16ビットもあれば不足はないでしょう。
8ビットCPUだけでなく16ビットCPUでも32ビットCPUでも個人的に使うことができればパソコンということね。
価格が下がればそれができるのね。
グラフィックスや画像処理（アニメ的なもの）なら32ビットスーパーパソコンが必要かもしれません。

この本むずかしかった
でしょうか？

あなたがもし、中学生や高校
生ならそれが当り前です。
大学生でもスラスラ読めた人
はそういないと思います。

マンガの書き方もあるかもし
れませんが、なにしろ相手は
小さいとはいえコンピュータ
ですからね。むずかしいのが
当り前かもしれません。

ハードウェアの方は時々刻々
と改良が加えられ新しいもの
に置き変わっていきます。で
すから移り変わっていく部分
よりも本質的なことを理解す
ることが大切だと思うのです。

時代が16ビット、32ビット
に進みメモリの大容量化、
小型化、スピードアップが
進もうとも、ここに書いた
基本的なことは、真空管の
時代から変わっていないの
です。

では、ひとまず
これで失礼します。
ごきげんよう、
さようなら。

絵とき
8ビットマイコン CPU 大研究　定価 1500 円

昭和59年9月10日　初版発行

著者　笹倉正義
発行者　中村洋一郎
発行所　株式会社 日本実業出版社
東京都文京区本郷3丁目2番12号　〒113
☎代表 03(814)5161　振替 東京 7-25349
大阪市北区西天満6丁目8番1号　〒530
☎代表 06(362)6141

印刷所　壮光舎印刷株式会社
製本所　株式会社　若林製本工場

ISBN4-534-00962-3 C0055 ¥1500E